FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Gerador de padrões de vídeo UHD utilizando HDL (Verilog)

Júnio Duarte Lopes Parente

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador interno: João Paulo de Castro Canas Ferreira

Orientador externo: José Armando de Sá Dias

24 de junho de 2019

Resumo

Perante a exigência do mercado e a crescente evolução tecnológica, os sistemas de vídeo estão a ser desenvolvidos para suportar resoluções UHD, que se demonstram mais apelativas para o consumidor final.

A interface digital série (SDI) é um conjunto de normas para transmissão de vídeo digital. A norma SDI que atinge maior velocidade é a SDI-12G, que chega a 12 Gbit/s. No entanto, no caso da transmissão em 8K 60 fps através da interface digital série é necessário transportar 48 Gbit/s, devido à quantidade de informação característica desta resolução.

Uma boa forma de testar uma interface de vídeo é através da utilização de um padrão de vídeo. No sentido de resolver o problema descrito relativo à quantidade de dados a transmitir, a presente dissertação consiste na implementação, em FPGA, recorrendo a HDL (Verilog) e IP Cores existentes, de um gerador de padrões de vídeo para altas resoluções como é o caso do 4K e do 8K, a 60 *frames* por segundo (fps). As *frames* geradas pelo gerador de padrões de vídeo, são enviadas para a interface SDI e são utilizadas técnicas *multilink* para se conseguir enviar uma cadência de dados mais elevada que não conseguiria ser transmitida através de um único *link* SDI. Os padrões gerados são compostos por um fundo selecionável, por um objeto intermitente e por um *timecode*.

Este projeto foi desenvolvido para a MOG Tecnologies, com o intuito de ajudar outros projetos que, neste momento estão a ser desenvolvidos na empresa, também relativos à transmissão de vídeo 8K 60 fps com recurso à interface SDI 12G utilizando *multilink*.

Abstract

Faced with market demand and increasing technological evolution, video systems are being developed to support UHD resolutions, which are proving more appealing to the final consumer.

The serial digital interface (SDI) is a set of standards for digital video transmission. The highest speeding SDI currently is 12G SDI, which reaches 12 Gbit/s. However, in case of transmission at 8K 60 fps via the serial digital interface, 48 Gbit/s must be carried due to the amount of information characteristic of this resolution. A good way to test a video interface is through the use of a video pattern.

In order to solve the described problem regarding the amount of data to be transmitted, this dissertation consists in the implementation, in FPGA, using HDL (Verilog) and existing IP Core, of a video pattern generator for high resolutions such as 4K and 8K, at 60 frames per second (fps). The generated frames by the video pattern generator are sent to the SDI interface and multi-link techniques are used to send a higher data rate that could not be transmitted via a single SDI link. The generated patterns are composed of a selectable background, a intermittent object and a timecode.

This project was developed for MOG Technologies, in order to help other projects that are currently being developed in the company, also related to the transmission of video 8K 60 fps using the interface 12G SDI multilink.

Agradecimentos

À minha família, especialmente ao meu pai, mãe, irmão e avós que sempre me apoiaram na minha vida académica.

À minha namorada Débora que me acompanhou em todos os momentos dando-me sempre motivação para continuar.

Ao meu orientador João Canas Ferreira, por ter tido o privilégio de ser orientado por ele neste projeto e pelo acompanhamento técnico e científico.

Ao meu orientador na empresa José Dias por me ter dado a oportunidade de realizar a minha dissertação na MOG.

A todos os meus colegas da MOG, especialmente ao José Ribeiro, Nuno Silva e José Meira por estarem sempre disponíveis para me ajudar.

A todos os meus colegas da FEUP que trabalharam a meu lado e me motivaram para desenvolver este projeto. Especialmente ao Daniel Gonçalves, Paulo Correia e João Afonso que me apoiaram com a sua amizade nos últimos anos.

Ao meu amigo Ruben Costa pela motivação por sempre acreditar em mim e por apoiar as minhas escolhas.

E por último, mas não menos importante, ao Professor José Leme que me ajudou a descobrir o caminho e me motivou a seguir Engenharia Eletrotécnica.

Júnio Duarte Lopes Parente

"Good. Let's get it"

Sekouba Conde

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

1080p	Resolução com a dimensão de 1920 por 1080 pixeis
2SI	*Two-sample interleave*
4k	Resolução com a dimensão de 7680 por 2160 pixeis
8k	Resolução com a dimensão de 3840 por 4320 pixeis
AXI	*Advanced extensible interface*
BCN	*Bayonet Neill–Concelman*
BRAMS	*Block RAMs*
DDR	*Dual data rate*
DSP	*Digital signal processing*
FIFO	*first in, first out*
FF	*flip flops*
FPGA	*Field-programmable gate array*
fps	*frames per second*
Gb	Gigabit
GB	Gigabyte
Gbit/s	Giga *bits* por segundo
GHz	Giga Hertz
GT	*Gigabit transceivers*
HDL	*Hardware Description Language*
HDMI	*High Definition Multimedia Interface*
HD	*Higth Defenition*
HTTP	*Hypertext Transfer Protocol*
Hz	Hertz
HVC	*Scalable High Efficiency Video Coding*
I2C	*Inter-Integrated Circuit*
ILA	*Integrated Logic Analyzer*
IP	*Internet Protocol*
IP core	*Intellectual property core*
LUTS	*Look Up Tables*
Mb	Megabit
MB	Megabyte
MHz	Mega Hertz
MT	*Mega transfers*
MMT	MPEG *media trasnport*
MPEG	*Moving Picture Experts Group*
Mux	Multiplexador
QSFP	*Quad Small Form-factor Pluggable*

RAM	*Random Access Memory*
RDIMM	*Registered dual in-line memory module*
RGB	R- *red* G- *green* B-*blue*
SDI	*Serial Digital Interface*
SMPTE	*Society of Motion Picture and Television Engineers*
SOF	*Start of frame*
YCbCr (YUV)	Y- Luminância Cb(U)- Crominância do Azul Cr(V)- Cominância do vermelho (no digital)
UHD	*Ultra Higth Defenition*
VDMA	*Video Direct Memory Access*

Capítulo 1

Introdução

1.1 Contexto

A evolução tecnológica tem vindo a permitir a utilização de resoluções de vídeo cada vez mais avançadas enquadradas no conceito de alta definição (desde HD (*High Definition*), a *Full*-HD e mais recentemente UHD (*Ultra High Definition*)).

A resolução 4K enquadra-se no conceito de UHD e surgiu na indústria cinematográfica há cerca de 10 anos alargado-se para dispositivos do nosso dia a dia como é o caso das televisões e telemóveis [1]. Mais recentemente surgiu o 8K, motivado pela utilização de resoluções cada vez maiores, mais apelativas e desafiantes. Esta apresenta 4 vezes mais pixeis por *frame* que o 4K, também se enquadra na categoria de UHD e recentemente o seu transporte e distribuição têm vindo a evoluir de uma maneira muito positiva nos últimos anos[2].

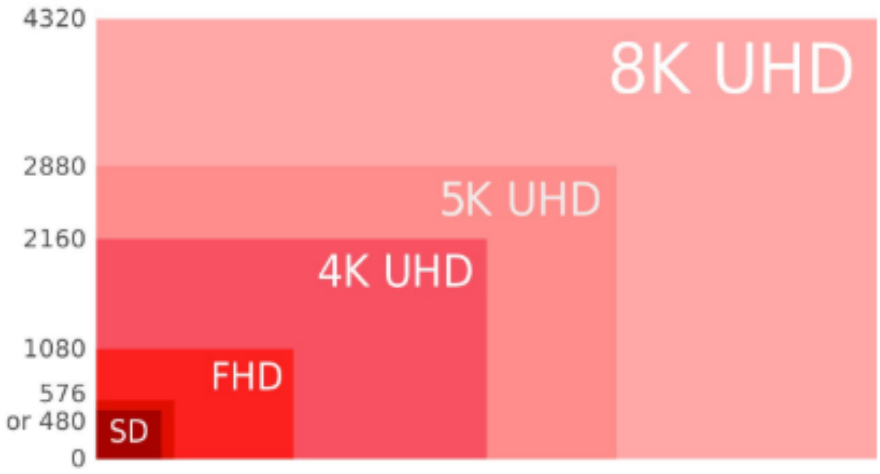


Figura 1.1: Rápida evolução das resoluções na ultima década. Fonte:[3].

Para atender às necessidades e maior exigência por parte do setor audiovisual e multimédia (*broadcasters*, criadores e distribuidores de conteúdo audiovisuais, entre outros), hoje em dia existe uma pressão crescente no grau exigência e qualidade no sentido de criar resoluções mais

apelativas para o consumidor final. Em consequência, surgem novos desafios tecnológicos envolvidos no processo, destacando-se a capacidade de processamento em tempo real devido à quantidade de informação que é necessária processar para as resoluções em questão. Este é portanto, um desafio aliciante frente às possibilidades de evolução deste mercado e do seu padrão de exigências [4].

1.2 Motivação

Pelos motivos referidos anteriormente na contextualização do problema, a transição do transporte e distribuição de vídeo recorrendo a formatos tradicionais de SDI (*Serial Digital Interface*) para o transporte e distribuição de vídeo através de IP (*Internet Protocol*) tem gerado um crescente entusiasmo junto da indústria de difusão, motivado pelos avanços registados a nível das redes IP. As redes IP assumem-se, cada vez mais, como uma alternativa capaz de manter as propriedades do vídeo, mesmo no transporte/distribuição em longas distâncias [5].

Mas apesar das evoluções das referidas redes, não é viável a sua utilização em exclusivo pelos seguintes motivos:

- Não estarem suficientemente desenvolvidas para atingir um desempenho satisfatório [6];
- Apresentarem problemas de interoperabilidade de formatos e inexistência de *standards* unificados [6];
- A generalidade de sistemas de *broadcast* assentarem essencialmente na interface SDI [2];
- As tecnologias SDI terem sofrido também evoluções, existindo experiências favoráveis associadas à utilização da interface SDI 12G para o transporte e distribuição de vídeo de altas definições [6].

As redes IP e a interface SDI não devem ser vistas como concorrentes, mas sim como complementares. Na Figura 1.2, é possível observar uma instalação de produção de media audiovisual genérica com um funcionamento híbrido em que a comutação em tempo real é realizada através de SDI e a comutação que não precisa de ser realizada em tempo real (transferência de ficheiros) utiliza IP [2].

A utilização de padrões de vídeo é uma das formas mais simples e eficazes de testar uma interface de vídeo. A construção e injeção de um padrão de vídeo conhecido numa extremidade da cadeia e a comparação da saída da cadeia com o mesmo padrão que foi injetado, permite retirar conclusões relativas ao funcionamento da interface. Se as duas imagens forem iguais, a interface está a funcionar como se esperava. Se forem diferentes, é possível que a interface não esteja a funcionar como o esperado e permite detetar pistas que podem ajudar a resolver problemas.

A interface SDI 12G, utilizada para a transmissão e distribuição de vídeo em 4k, também permite transmitir em 8K através da utilização de uma interface com 4 *links* SDI 12G coordenados e sincronizados (*multilink)*. O estudo da utilização de *multilink* 4 x 12G SDI para resoluções 8K

revela-se importante na indústria deste setor, e mais propriamente para a empresa MOG Technologies, pois têm o objetivo de ajudar o desenvolvimento de outras plataformas que, neste momento, estão em projeto na empresa. É com o desígnio de complementar a investigação em torno das tecnologias referidas anteriormente, estabelecendo padrões para o teste das interfaces SDI 12G, que surge o projeto cujo desenvolvimento é descrito nesta dissertação.

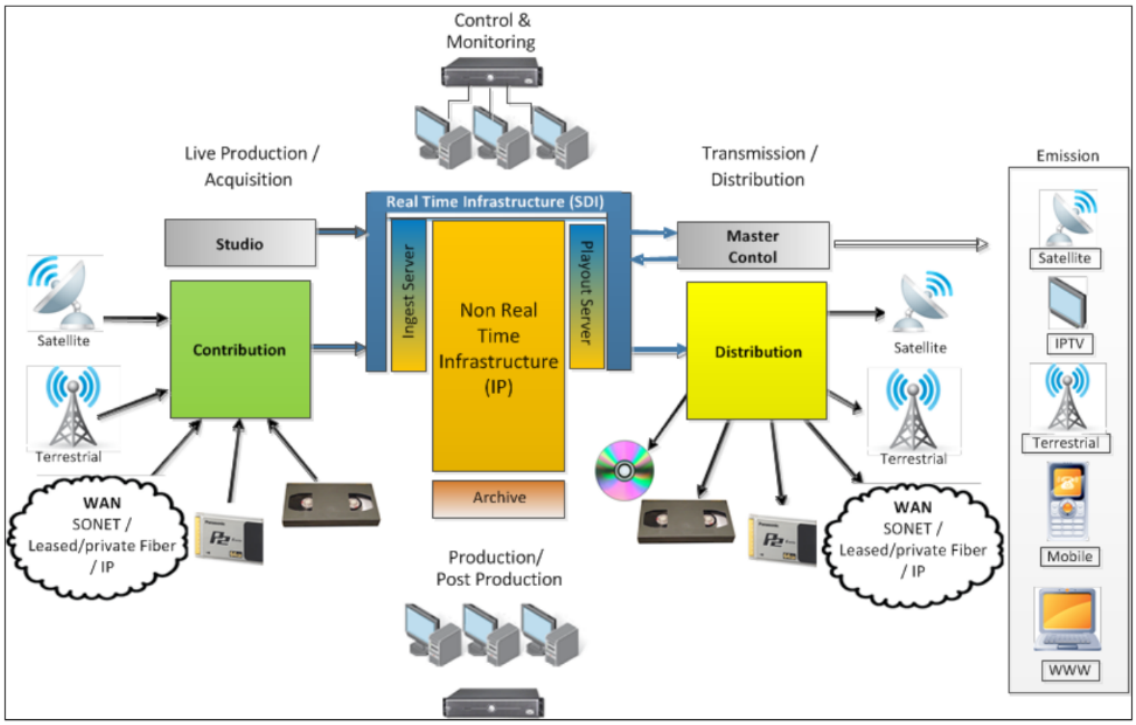


Figura 1.2: Instalação de produção de mídia híbrida audiovisual genérica. Fonte: [2].

1.3 Objetivos

O principal objetivo deste trabalho é a implementação em FPGA, recorrendo a HDL (*Verilog*) e módulos IP existentes, de um gerador de padrões de vídeo UHD (de 8K), a 60 *frames* por segundo (*fps*), em contexto de *broadcasting*, capaz de ser utilizado para testar a interface SDI.

Os objetivos específicos são:

- Análise do estado da arte relacionado com o transporte/distribuição de vídeo (em especial, vídeo UHD) através de SDI (com particular destaque para SDI 12G);
- Implementação em FPGA de um gerador de sinal de vídeo UHD (8K) capaz de gerar múltiplos padrões;
- Desenvolvimento de um plano de teste para o sistema;

- Aplicação de elementos gráficos e *timecodes* sobre os padrões de maneira a criar um teste mais completo;
- Dividir as *frames* para poderem ser enviadas por 4 interfaces SDI 12G;
- Realização de um *testbench* para analisar as imagens antes de serem enviadas para a interface SDI em fase de simulação;
- Validação através de simulação e teste em condições reais.

Este último objetivo, está dependente de outros projetos que estão a ser desenvolvidos neste momento na empresa do lado do recetor SDI, que podem não estar finalizados a tempo.

O conjunto de módulos que foi implementado pode ser visto com um "*testbench* sintetizável", pois trata-se de um circuito sintetizável que irá permitir o teste das interfaces SDI utilizadas em produtos desenvolvidos pela empresa.

1.4 Estrutura do documento

Esta dissertação está organizada da seguinte forma:

- Capítulo 2: Enquadramento Técnico e Estado da Arte: Este capítulo introduz alguns conceitos teóricos relativos a este projeto.
- Capítulo 3: Estudo do sistema a desenvolver: Neste capítulo é apresentada a análise de requisitos, a arquitetura, o plano de testes para o sistema e a divisão do projeto segundo 4 subsistemas principais.
- Capítulo 4: Subsistema que configura o gerador de padrões de vídeo: Este capítulo apresenta a implementação, teste e resultados deste subsistema.
- Capítulo 5: Subsistema que gera o objeto intermitente: Este capítulo apresenta a implementação, teste e resultados deste subsistema.
- Capítulo 6: Subsistema que gera o *Timecode*: Este capítulo apresenta a implementação, teste e resultados deste subsistema.
- Capítulo 7: Subsistema que implementa realiza a divisão da imagem por 4 *links*: Este capítulo apresenta a implementação, teste e resultados deste subsistema.
- Capítulo 8: Sistema completo: Integração dos 4 subsistemas com os módulos característicos da FPGA e com os responsáveis pelo envio para os *links* SDI. Teste e resultados para o sistema completo.
- Capítulo 9: Conclusões: Por último são apresentadas conclusões relativas ao projeto e o trabalho futuro.

Capítulo 2

Enquadramento Técnico e Estado da Arte

Neste capítulo, são apresentados vários tópicos referentes aos fundamentos e trabalhos realizados nos quais se baseia este projeto. São abordadas temáticas como formatos e especificações de vídeo não comprimido, transporte de vídeo e interfaces de vídeo.

2.1 Padrão de vídeo

Um padrão de vídeo consiste num conjunto de *frames* padronizadas que são utilizadas para diversos fins como realizar uma calibração confiável e eficiente, testar e solucionar problemas em *displays* de vídeo e também para testar interfaces vídeo (SDI, HDMI, etc). Na figura 2.1, é possível observar alguns dos padrões de vídeo mais utilizados na indústria.

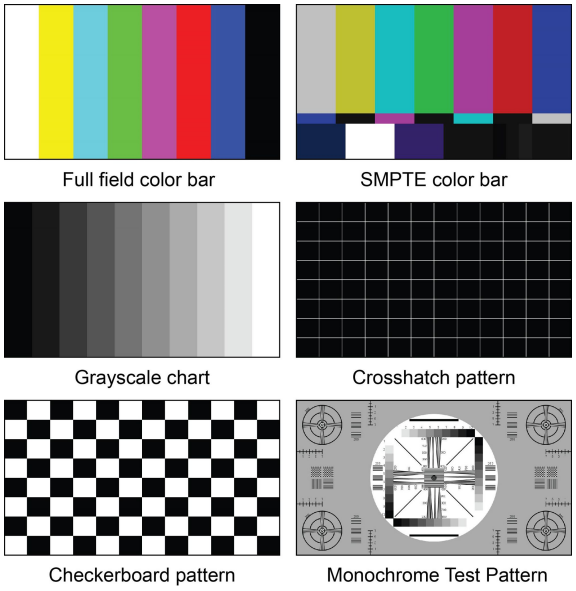


Figura 2.1: Padrões de vídeo. Fonte: [7].

2.2 Características de vídeo

Nesta secção são abordados alguns conceitos importantes a ter em conta quando se realiza um projeto na área de distribuição de vídeo, como é o caso da resolução, *aspect Ratio*, *frame rate* e etc.

2.2.1 Resolução

A resolução de vídeo descreve a quantidade de detalhe de uma imagem. O termo resolução é normalmente utilizado para quantificar o número de pixeis de uma imagem. Portanto, a resolução em pixeis é descrita como um conjunto de dois números positivos, em que o primeiro simboliza o número de colunas (largura) e o segundo o número de linhas (altura) [8]. Na figura abaixo é apresentado um exemplo de como uma imagem pode aparecer dependendo da resolução em pixeis.

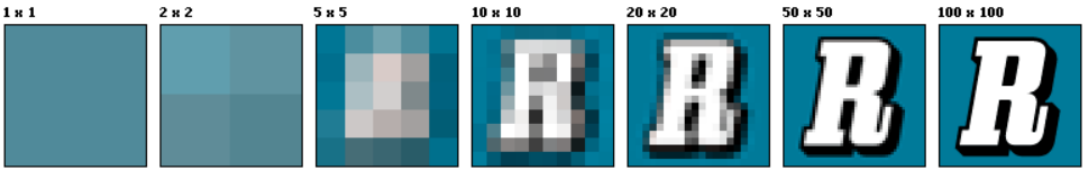


Figura 2.2: Diferentes resoluções em número de pixeis por *frame*. Fonte: [8].

2.2.2 *Aspect Ratio*

O *aspect ratio* é relação entre a altura e a largura de uma imagem. Os dois *aspect ratios* mais conhecidos são o 4:3 (também chamado de 1.33) e o 16:9 (também chamado de 1.78) [9]. Por exemplo, uma resolução 1920 x 1080 apresenta o *aspect ratio* de 16:9.

2.2.3 *Frame Rate*

Antes de introduzir o conceito de *frame* rate, vai ser introduzida a noção de *timecode*. Este é um código que faz referência a uma determinada *frame* de vídeo num determinado instante, ou seja, cada *frame* de uma *stream* de vídeo têm o seu *timecode* associado [10]. Na figura 2.3 é possível observar um exemplo de um *timecode* da *frame* número 6, do segundo 20, do minuto 53, da hora 18 de um vídeo.

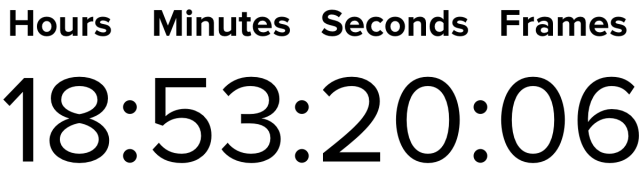


Figura 2.3: Exemplo de *Timecode*. Fonte: [10].

O *frame rate* é a grandeza que indica o número de *frames* por segundo que compõem o vídeo. Mas, existem dois tipos de *frame rates*, o inteiro (30, 60 *fps* por exemplo) e o fracionário (29.97, 59.94 *fps* por exemplo).

Como num vídeo se pretende que o *timecode* esteja síncrono com o tempo real, foi criado o *timecode* fracionário para utilizar em vídeos com *frame rates* fracionários. Num *frame rate* fracionário existe menos uma *frame* por cada 1000 do que num *frame rate* inteiro no intervalo de transmissão de 1000 *frames*. No caso de 30 *fps* fracionário (29.97) existem menos 108 *frames* por hora (30 *frames* x 60 segundos x 60 minutos x 1/1000 = 108) do que no caso dos 30 *fps* inteiros. Para este caso, em todos os minutos com exceção dos múltiplos de 10, o relógio, no inicio de cada minuto apresenta o valor 2 na parte que conta as *frames*, descartando assim o número 0 e 1 para compensar o valor presente no *timecode* relativamente ao tempo real do vídeo. Isto grante que em 54 dos 60 minutos exista um salto no contador de duas unidades para compensar as 108 *frames* a menos.

Resumindo, o *timecode* é um contador de *frames* que, no caso de estarmos perante um *frame rate* fracionário, em certos instantes predefinidos de tempo, dá um salto para um valor mais à frente de forma a manter o sincronismo com o tempo real.

2.2.4 Vídeo progressivo e entrelaçado

No vídeo progressivo, cada *frame* é capturada de uma só vez e contém toda a informação da imagem. Nos tempos correntes, este tipo de vídeo é o mais utilizado para visualização em monitores digitais [11].

Por sua vez, no vídeo entrelaçado cada *frame* é composta por dois campos obtidos em instantes distintos, cada um deles com metade da informação da *frame*. O instante de tempo entre as duas aquisições é igual a 1/*frame rate* segundos.

2.3 Interface SDI

O SDI é uma norma para transmissão de vídeo digital, através de cabo coaxial ou fibra ótica. É uma família de interfaces padronizada pelo SMPTE (Sociedade de Engenheiros de Cinema e Televisão). O fluxo de dados é serializado e os dados não são comprimidos antes de transmitir. A

norma SDI utiliza palavras com tamanho de 8 ou 10 bit e é utilizada para transmitir *streams* de dados descompactados e sincronizados entre o transmissor e o recetor [12].

A utilização de uma interface de vídeo não comprimida neste domínio deve-se ao facto de, a adoção de uma interface comprimida ser impraticável num estúdio, pois para tal era necessário equipamento de monitorização, sistemas de compressão e de descompressão de dados. Consequentemente, tal facto seria um problema acrescido devido à diversidade de sistemas de compressão e formato de dados, e à perda de qualidade da imagem.

Relativamente à interface eletrica, os cabos coaxiais utilizados para transmissão recorrendo a SDI apresentam uma impedância de 75 Ω. São também utilizados conectores BNC (*Bayonet Neill-Concelman*) para fazer as interligações. O comprimento máximo dos cabos varia com a velocidade máxima dos dados a transmitir. Para SDI 12 G, que é a norma utilizada para o 8K *multilink*, o comprimento máximo é de 60 metros. No anexo A, é apresentada uma tabela com mais detalhes relativos às diferentes normas [13].

2.3.1 Normas SDI

Para além dos dados relativos aos pixeis da imagem, também existem os dados auxiliares que não contêm informação relativa aos pixeis da imagem, mas são fulcrais para uma correta interpretação dos dados por parte do recetor. É o caso dos metadados, que em HD estão normalizados pela norma STMPE ST-352. Na tabela 2.1, estão descritas as diferentes normas para as diferentes velocidades de transmissão de dados existentes em SDI.

Tabela 2.1: Normas para diferentes velocidades de dados SDI. Fonte: [14].

Norma	**Nome**	**Ano**	***Bitrate***
SMPTE 259M	SD-SDI	1989	270 Mbit/s
SMPTE 344M	ED-SDI		540 Mbit/s
SMPTE 292M	HD-SDI	1998	1.5 Gbit/s
SMPTE 372M	Dual Link HD-SDI	2002	3 Gbit/s
SMPTE 424M	3G-SDIS	2006	3 Gbit/s
SMPTE ST-2081	6G-SDIS	2015	6 Gbit/s
SMPTE ST-2082	12G-SDIS	2015	12 Gbit/s
SMPTE ST-2083	24G-SDIS	Em desenvolvimento	24 Gbit/s

2.3.2 Utilização de vários *links* de dados (*Multilink*)

A transmissão de dados utilizando SDI pode ser realizada através de diferentes formas, com apenas uma interface ou com várias (*multilink*). Quando se pretende uma cadência de dados mais elevada, em vez de utilizar apenas uma interface que permita essa cadência de dados, pode utilizar-se um número de interfaces com cadência de dados inferior, que em conjunto fornecem a cadência

de dados pretendida. Isto é, nada mais nada menos, do que utilizar *multilink*, ou seja, utilizar vários *links* de dados [15].

Por exemplo, para obter uma cadência de dados de 12 Gbit/s pode utilizar-se:

- Uma interface 12G-SDI;
- Duas interfaces 6G-SDI (*dual-link)*;
- Quatro interfaces 3G-SDI (*quad-link)*.

O estudo deste tipo de tecnologia revela-se importante, pois ainda não existe uma interface SDI com velocidade suficiente para transmitir 8K a 60 *fps* com apenas um *link*. No entanto, é possível transmitir 4K a 60 *fps* com um único *link* 12G-SDI. Como já foi referido, o 8K apresenta 4 vezes mais pixeis para serem transmitidos e uma forma de atingir o objetivo de utilizar 8K a 60 *fps* é utilizar 4 *links* de dados 12G-SDI.

A tecnologia *multilink* comparada com a utilização de apenas um *link* para uma dada cadência de dados pretendida, apresenta a desvantagem de utilizar um maior número de cabos, aumentando assim os custos. Por outro lado, o comprimento máximo dos cabos também é mais longo. Um cabo 12G-SDI tem um comprimento máximo de 60 metros, enquanto que, um cabo 3G-SDI pode ir até aos 200 metros.

Figura 2.4: Diferente número de cabos para a velocidade 12 Gbit/s. Fonte: [16].

Antes de existir a interface 12G-SDI, também foram utilizadas técnicas de *multilink* para se transmitir em 4K, o que prova que a utilização de *multilink* pode ser uma boa abordagem para o problema da elevada cadência de dados necessária para atingir o 8K [15].

2.3.3 Utilização de 12G-SDI

A norma 12G-SDI foi desenvolvida para suportar melhores resoluções, melhores taxas de dados, e melhor fidelidade de cores. Proporciona quatro vezes mais largura de banda do que o HD, suportando 12 Gbit/s, o que o torna ideal para o formato 4K a 60 *fps*. Foi introduzido em 2015 e a sua norma é a SMPTE ST-2082 [14].

A norma 24G-SDI está a ser desenvolvida para suportar diretamente o 8K. Mesmo suportando 8K com apenas um *link*, não é capaz de atingir os 60 *fps*, apenas atingirá os 30 *fps*. Estes motivos também contribuem para o estudo da utilização 4 interfaces 12G SDI para realizar a transmissão em 8K.

2.4 Formato - vídeo descomprimido

Os *codecs* de vídeo são algoritmos utilizados para codificar e descodificar *streams* de vídeo. A sua função é comprimi-los para enviar com um tamanho mais pequeno e descomprimi-los quando chegam ao seu destino. Alguns exemplos de *codecs* são o MPEG-1, o MPGE-2, o MPGE-4, e o H.206 [5] [17]. Quanto aos *containers* de vídeo, estes estão associados ao formato do vídeo. Os *containers* contêm as várias componentes do vídeo (*stream* de imagens, som, legendas e tudo o que seja necessário para a correta interpretação do vídeo). Alguns *containers* populares são por exemplo o 3GP, MPEG, AVI, OGG, MKV e FLV [17] [5].

O formato SDI, como já foi referido anteriormente, é um formato de vídeo descomprimido. Não utiliza *codecs* para comprimir informação.

2.5 Métodos de *multilink* - 2SI *vs square division*

Como já foi referido no capítulo 1, a técnica de *multilink* também é utilizada para transmitir em 4K. Existem diversos estudos que comparam a utilização do método 2SI com o *square division* para vídeo 4K transmitido através de *multilink* [18] [19] [20]. Na presente secção, estes métodos são analisados com recurso a exemplos para resoluções 4K construídas a partir de 4 *frames Full - HD*.

Tanto o método 2SI como o método *square division* decompõem uma *frame* original em 4 *frames* com um quarto da resolução, antes da informação entrar na cadeia de dados SDI. Do outro lado da cadeia, a imagem final é reconstruída.

2.5.1 *Square division*

O método *square division* consiste em dividir em quadrantes as imagem que se pretendem transmitir resultando 4 sub-imagens, todas com o mesmo número de pixeis (1/4 dos pixeis da original). Cada sub-imagem resultante é encaminhada para cada uma das 4 interfaces SDI. Na figura 2.5, pode observar-se um exemplo do resultado deste método, em que o quadrante *n* (de 1 a 4) tem de ser enviado pelo *n link* SDI [20].

Figura 2.5: Método *square division* para uma resolução 4K. Fonte: [20].

Esta abordagem apresenta duas desvantagens [18]. No caso de se perder um *link*, um quarto do imagem vai aparecer a negro. É também necessária muita memória comparando com o método 2SI do lado do recetor para guardar os 4 quadrantes e posteriormente apresentar a imagem corretamente.

Por outro lado, apresenta a vantagem de ser mais fácil de implementar.

2.5.2 Método 2SI

O método 2SI é completamente diferente do anterior e consiste em gerar 4 sub-imagens com 1/4 da resolução, ou colocar os 2 primeiros pixeis da primeira linha da imagem original no *link* 1, depois os dois seguintes no *link* 2 e repetir este processo até ao final da linha. A linha seguinte da imagem original sofre o mesmo processo mas agora para os *links* 3 e 4. Depois, é repetido o procedimento indicado até se ter percorrido todas as linhas da imagem original [18]. Na figura 2.6 é demonstrada a forma como se processa este algoritmo e na 2.7 é possível observar o aspeto visual das 4 *frames* resultantes. No fundo são 4 imagens com 1/4 da resolução da imagem original.

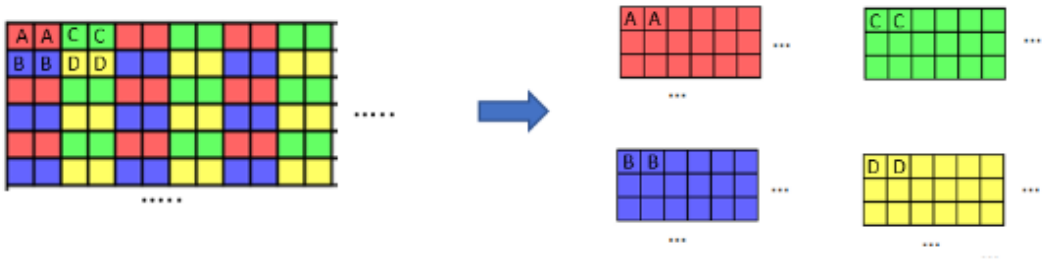


Figura 2.6: Divisão da imagem segundo 2SI (*link* 1 - vermelho, *link* 2 - verde, *link* 3 - azul, *link* 4 - amarelo).

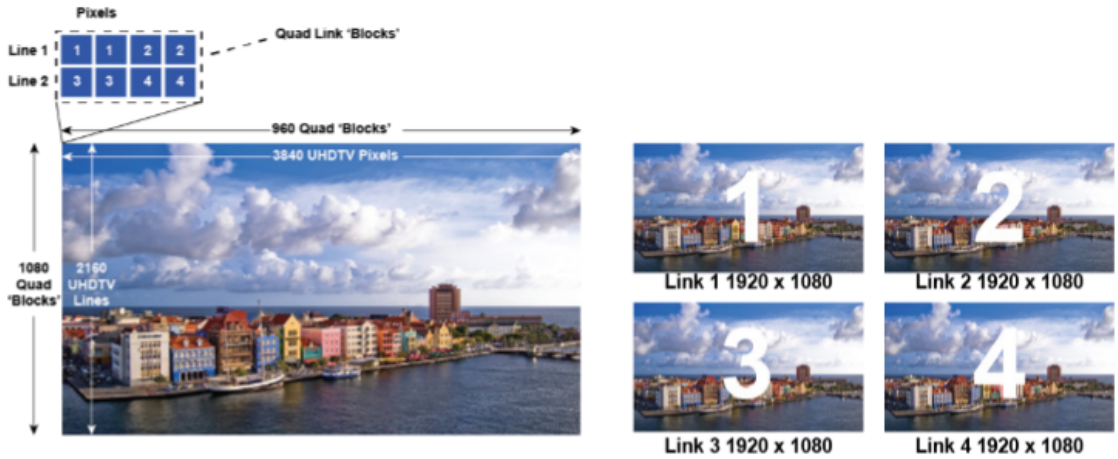


Figura 2.7: Método 2SI para uma resolução 4K. Fonte: [20].

A utilização do método 2SI permite uma configuração de memória mais simples, apresenta menos latência de processamento e é a técnica mais utilizada no mercado.

No entanto, existe perda de resolução na falha de um *link* e é mais difícil de implementar.

2.6 Transporte de vídeo em SDI

Num formato de vídeo cada pixel é representado normalmente por 8, 10, ou 12 bit, resultando um espetro de cores com $2^8 = 256$, $2^{10} = 1024$ e $2^{12} = 4096$ diferentes tonalidades, respetivamente.

Uma forma de representar cada pixel de uma *frame* é no formato RGB em que o R simboliza a componente vermelha, o G a verde e o B a azul. Existem vários tipos de RGB, sendo que o RGB 888 é um dos mais utilizados. Este formato apresenta um espetro de cores com 256 tonalidades para cada uma das componentes e necessita de 24 bit para guardar a informação referente a cada pixel (8 bit para cada componente).

Pesquisas médicas provaram que o olho humano é sensível, não só a cor mas também a brilho. Tendo em conta a sensibilidade do olho à cor e ao brilho, torna-se possível sub-amostrar algumas componentes da cor da imagem sem perda da qualidade para observação pelo olho humano [21] [22]. A utilização de formatos sensíveis à cor e ao brilho, como é o caso do YCbCr, é muito mais eficaz, pois utiliza estas duas componentes: crominância e luminância. O Y refere-se à luminância, enquanto Cb refere-se ao componente crominância do azul e Cr à componente crominância da cor vermelha. No anexo B, estão apresentadas as equações genéricas para a conversão de RGB para o formato YCbCr [23]. De maneira a melhor compreender a figura 2.8 é agora introduzido o conceito de YUV, que é nada mais nada menos, do que a representação do YCbCr no mundo analógico. Nesta figura é possível analisar a cor resultante para um dado conjunto de valores (U;V) com Y=0.5.

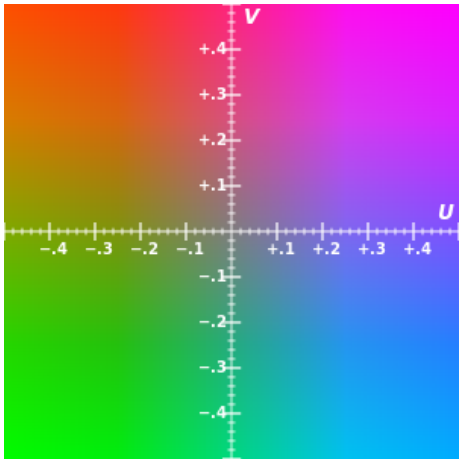


Figura 2.8: Plano de cores U-V com Y=0.5. Fonte: [24].

Na figura 2.9, estão representados algumas variantes do formato YCbCr como é o caso do YCbCr 4:2:2 10-bit, que é proposto pela MOG para realizar este projeto. Os dois componentes de

crominância são amostrados a metade da taxa de amostragem de luminância. Ou seja, a resolução de crominância horizontal é reduzida para metade, o que reduz a largura de banda de um sinal de vídeo não comprimido num terço, com pouca diferença visual.

Portanto, o formato YCbCr 4:2:2 10-bit utiliza 40 bit por cada dois pixeis da imagem:

- 20 para a luminância (10 para cada pixel);
- 10 para a crominância da cor azul (só é utilizada a do primeiro pixel de um grupo de 2 pixeis);
- 10 para a crominância da cor vermelha (só é utilizada a do primeiro pixel de um grupo de 2 pixeis).

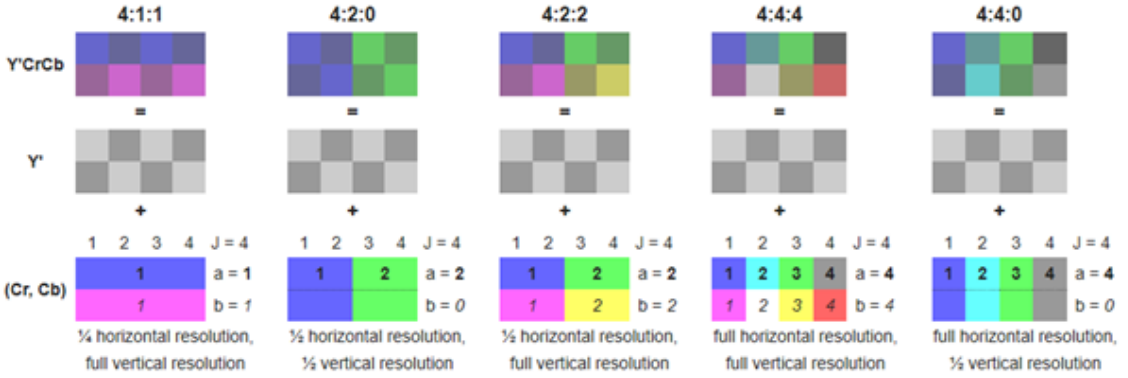


Figura 2.9: *Chroma subsampling*. Fonte: [25].

Na tabela 2.2, é possível verificar qual a velocidade necessária para transportar apenas o *payload* das diversas variantes do formato YCbCr para diferentes *fps* para a resolução 8K.

Horizontal Pixels	Vertical Pixels	Frames per Second (nominal)	Total Payload (nominal)					
			10-bit 4:2:0	10-bit 4:2:2	10-bit 4:4:4	12-bit 4:2:0	12-bit 4:2:2	12-bit 4:4:4
7680	4320	120	60Gbit/s	80Gbit/s	120Gbit/s	72Gbit/s	95.5Gbit/s	144Gbit/s
		60	30Gbit/s	40Gb/s	60Gbit/s	36Gbit/s	48Gbit/s	72Gbit/s
		50	25Gbit/s	33Gbit/s	50Gbit/s	30Gbit/s	40Gbit/s	60Gbit/s
		30	15Gbit/s	20Gbit/s	30Gbit/s	18Gbit/s	24Gbit/s	36Gbit/s
		25	12.4Gbit/s	16,6Gbit/s	25Gbit/s	15Gbit/s	20Gbit/s	30Gbit/s
		24	12Gbit/s	16Gb/s	24Gbit/s	14.4Gbit/s	19 Gbit/s	29Gbit/s

Tabela 2.2: Velocidade necessária para diferentes formatos YCrCb. Fonte: [26].

2.7 Algumas normas SMPTE importantes

Na documentação da SMPTE, o SMPTE OVERVIEW [27] permite identificar quais os documentos a utilizar para o desenvolvimento de uma determinada aplicação.

Estas normas indicam como deve ser tratada a informação para ser transmitida em SDI, pois têm de obedecer a certos princípios.

2.7.1 SMPTE ST 2082-10

A norma SMPTE ST 2082-10 define o mapeamento de uma imagem para uma interface 12G SDI. Este processo está ilustrado na figura 2.10. Para realizar o mapeamento é utilizado o método 2SI gerando 4 sub-imagens cada uma contendo duas *streams* de dados. No final, as 8 *streams* são multiplexadas para uma saída 12G-SDI.

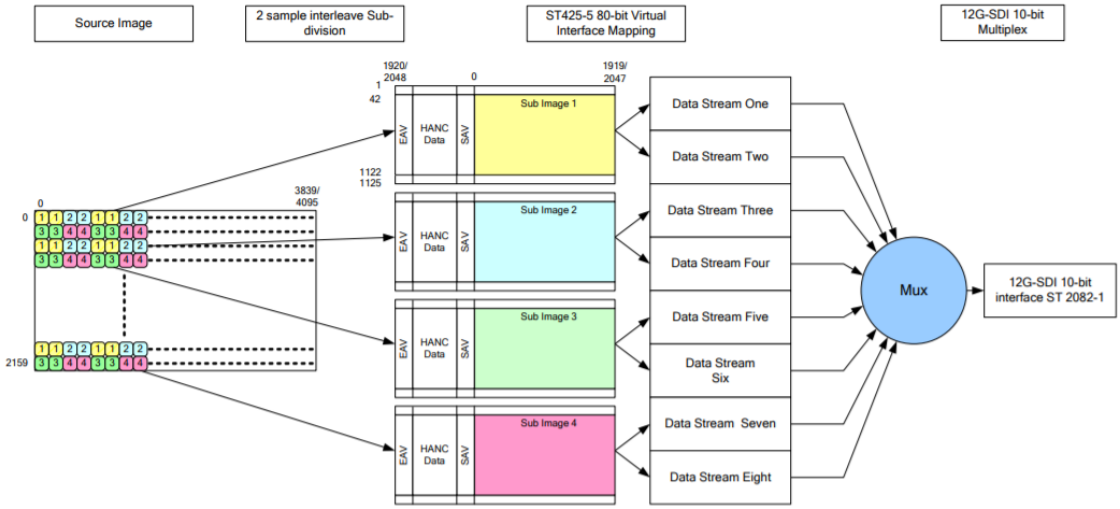


Figura 2.10: Mapeamento de uma *stream* 4K numa interface 12G-SDI [28].

2.7.2 SMPTE ST 2082-11

Para além de conter informações relativas à interface elétrica, este documento também apresenta mais alguma informação relativa à multiplexação de todos os dados pertencentes a uma *frame* (auxiliares e imagem). Na figura 2.11, é possível verificar a multiplexação das 8 *streams* de dados mencionadas no tópico anterior.

Na parte de cima da imagem estão representadas as 8 *streams* (duas da mesma cor para cada sub-imagem). Na parte de baixo está demonstrada a multiplexação dos dados, começando pelos primeiros 10 bit da *stream* 8 até ao 10 primeiros da *stream* 1, repetindo ente ciclo (da *stream* 8 para a 1) até ao final de uma *frame*.

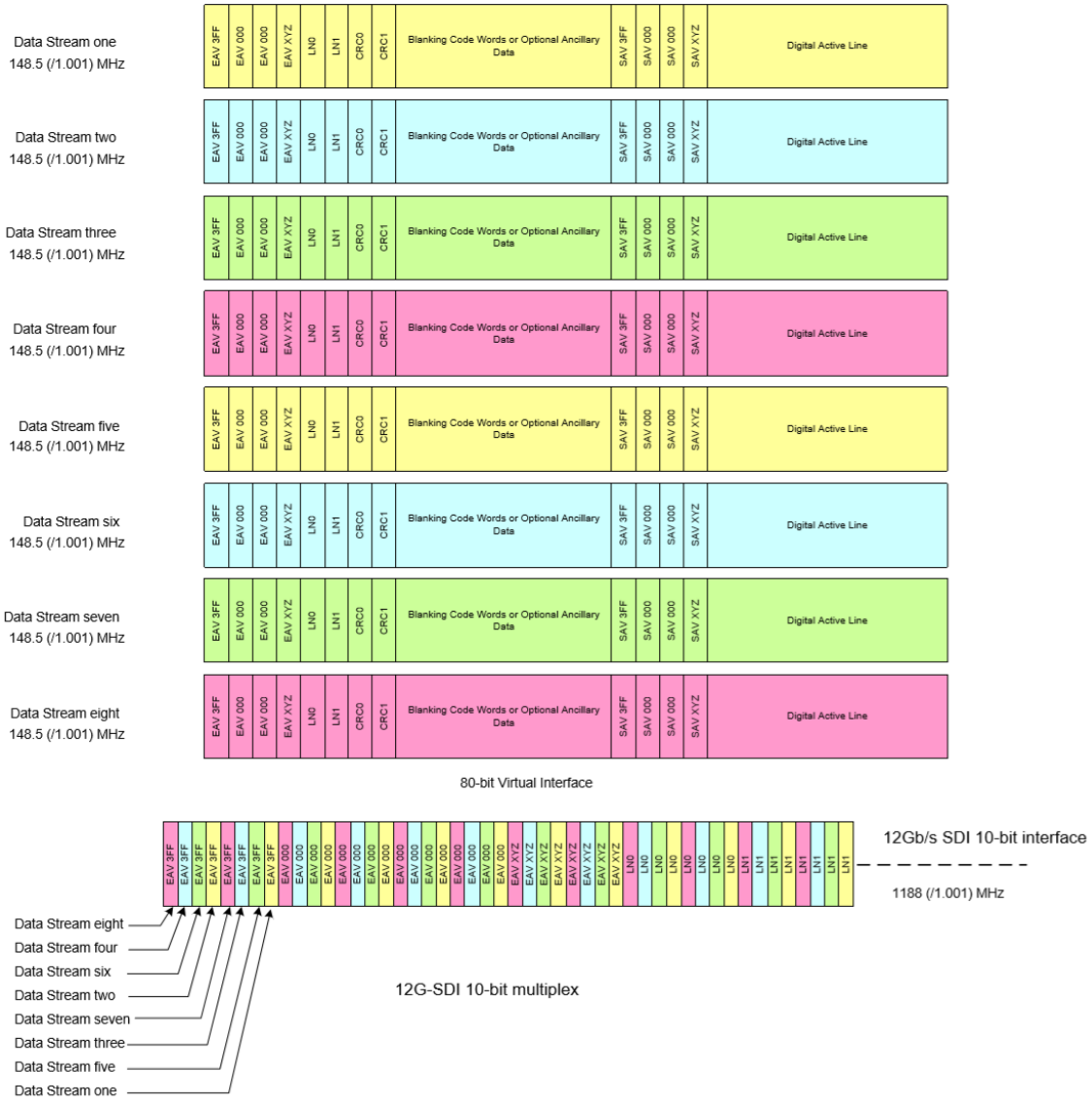


Figura 2.11: Mapeamento para 12G-SDI 10-bit com a informação auxiliar integrada . Fonte: [29].

2.7.3 SMPTE ST 2082-12

Esta norma da SMPTE define o mapeamento das imagens e da sua informação auxiliar para uma interface *quad-link* 12G SDI. O principal processo de transformação de uma imagem em 4 imagens para ser transmitida por 4 interfaces está ilustrado nas figuras 2.12 e na 2.13.

Após a divisão em 4 sub-imagens, estas são novamente divididas, mas agora em 16 sub-imagens, gerando 32 *data streams* que são multiplexadas recorrendo a 4 multiplexadores (1 para cada *link* de dados 12G SDI). As *data sreams* 1 a 8 vão par ao *link* 1, as 9 a 16 para o 2, 17 a 24 para o 3 e 25 a 32 para o 4. Este processo é realizado através do método 2SI.

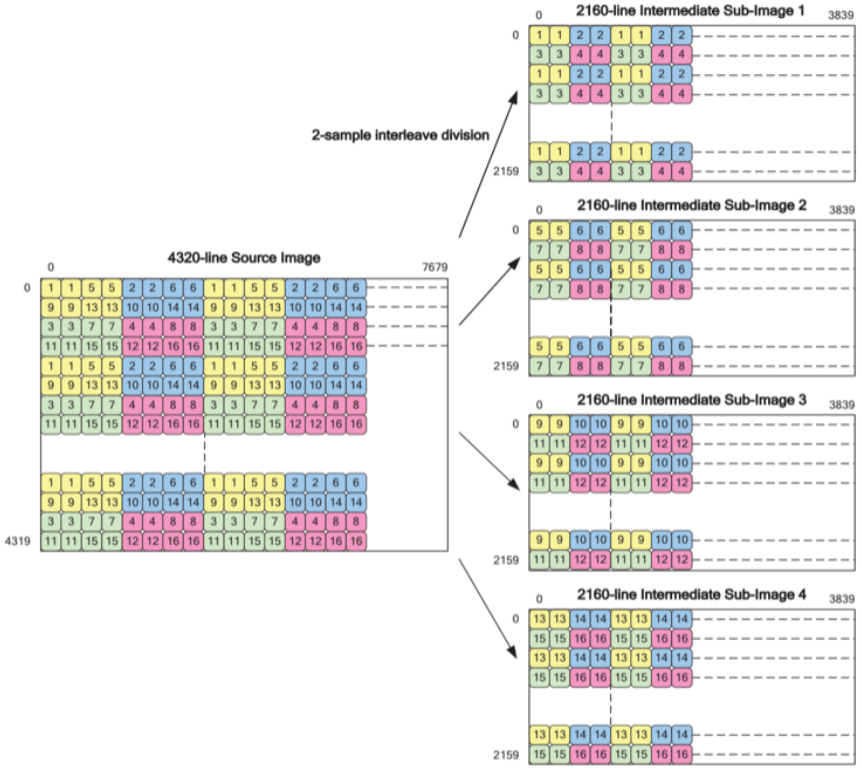


Figura 2.12: Divisão de uma imagem com 4320 linhas em uma com 2160 . Fonte: [30].

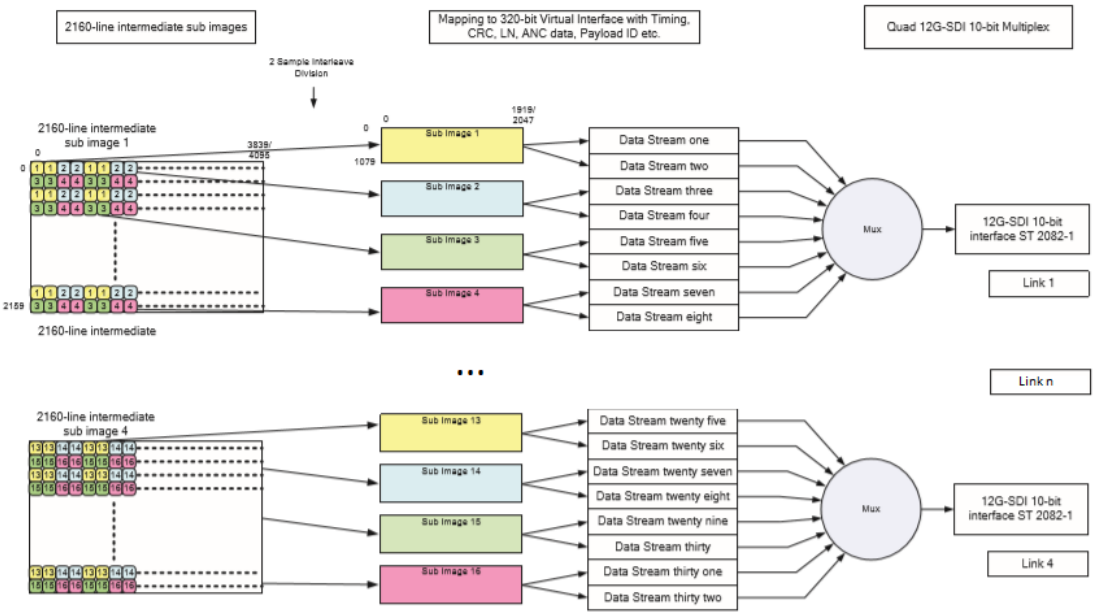
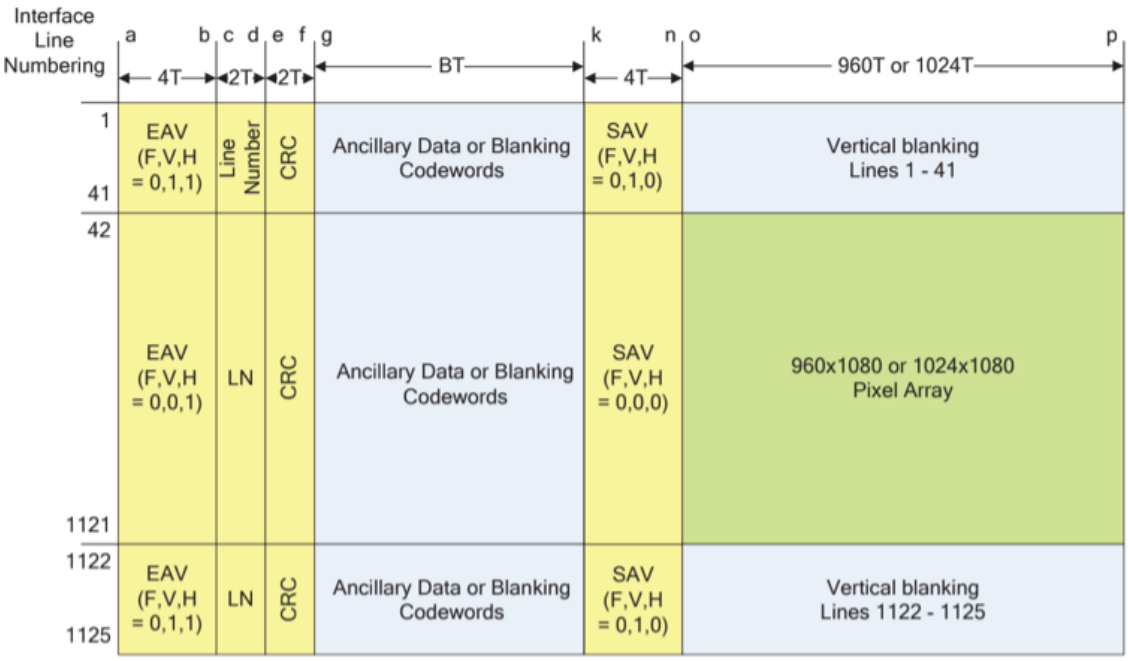


Figura 2.13: Mapeamento para para uma interface *quad-link* 12G SDI. Fonte: [30].

Como já foi referido, a *stream* de dados não apresenta só dados relativos à cor de cada pixel. Apresenta, ainda, dados auxiliares que também são divididos e, posteriormente, multiplexados. O que resulta em 16 *frames* iguais à da figura 2.14 que contém dados de cada uma das 16 sub-imagens e dados auxiliares. Nesta figura é possível observar um exemplo de uma *frame* com todas as suas componentes. A parte inferior da figura apresenta os tamanhos de cada componente da *frame* para as diferentes resoluções que podem ser utilizadas, ou seja, ilustra o número de bit de cada componente da imagem para uma determinada linha.

No anexo C, é possível verificar o mapeamento para *quad-link* 12G-SDI 4 x 10-bit com todas as componentes que compõem uma *frame*.

Na documentação SMPTE ST 2082-12 [30], existe informação mais detalhada sobre os dados auxiliares que é necessário transmitir.



Sub Image Format	a	b	c	d	e	f	g	BT	k	n	o	p
1920 x 1080 / 120	960	963	964	965	966	967	968	128	1096	1099	0	959
1920 x 1080 / 120/1.001	960	963	964	965	966	967	968	128	1096	1099	0	959
1920 x 1080 / 100	960	963	964	965	966	967	968	348	1316	1319	0	959
2048 x 1080 / 120	1024	1027	1028	1029	1030	1031	1032	64	1096	1099	0	1023
2048 x 1080 / 120/1.001	1024	1027	1028	1029	1030	1031	1032	64	1096	1099	0	1023
2048 x 1080 / 100	1024	1027	1028	1029	1030	1031	1032	284	1316	1319	0	1023
2048 x 1080 / 96	1024	1027	1028	1029	1030	1031	1032	339	1371	1374	0	1023
2048 x 1080 / 96/1.001	1024	1027	1028	1029	1030	1031	1032	339	1371	1374	0	1023

Figura 2.14: Estrutura de uma *data stream*. Fonte: [30].

2.8 Protocolos AXI

Os protocolos AXI são protocolos da especificação da interface AMBA que permitem a comunicação entre módulos independentes de uma solução ASIC. Estes facilitam o desenvolvimento de projetos com muitos controladores e periféricos com uma arquitetura de barramento.

Nesta secção, são apresentados dois protocolos *AXI* que são importantes para a realização deste projeto, o protocolo *AXI4-stream* e o *AXI4-lite*.

2.8.1 Protocolo *AXI4-Stream*

O protocolo *AXI4-stream* foi desenhado com o objetivo de transmitir *streams* de forma unidirecional entre um mestre e um escravo [31]. Este protocolo têm diversos sinais, mas os mais importantes para a realização deste trabalho são os seguintes:

Tabela 2.3: Principais sinais do protocolo *AXI-stream*.

Sinal	**Descrição**
Tready	É colocado a 1 quando o escravo está pronto para receber informação
Tuser (SOF)	Sinal que indica o inicio de uma *frame*
Tvalid	Está a 1 quando estão a ser transmitidos dados, simboliza que estes são válidos
Tlast	Indica o final de uma linha de dados
Tdata	Barramento em que são enviados os dados da *stream*

No protocolo *AXI4-stream* é tranferido o número de bit correspondentes ao tamanho de *Tdata* por ciclo de relógio quando *tvalid* é colocado a 1 após o escravo ter enviado a informação que está pronto para receber colocando *tready* a 1. Na figura 2.15 é possível observar as interações genéricas entre o mestre e o escavo. Por sua vez, na figura 2.16 é possível observar um exemplo desta interação em que o mestre inicia a transmissão no ciclo de relógio número 2 com o envio de um SOF (*start of frame*) e no ciclo seguinte o escravo responde com *ready* para dizer que está pronto para receber. Para poder enviar os dados através do barramento, no mesmo ciclo de relógio os sinais *valid* e *ready* têm de estar a 1. Na figura 2.16 é possível verificar que só são transmitidos dados no ciclo 6, 7, 8 e 11. Nos outros ciclos estes dois sinais não estão ambos 1. Por exemplo no flanco ascendente do ciclo de relógio 4,5 e 10 o *ready* não está a 1 e no flanco ascendente do ciclo 9 o *valid* não está a 1.

2.8.1.1 Protocolo *AXI4-Stream* para vídeo

O protocolo *AXI4-Stream* quando utilizado para transmitir uma *stream* de vídeo, apresenta duas particularidade adicionais:

- O sinal *Tlast* indica o último pixel de cada linha;
- O sinal *Tuser* indica o primeiro pixel da *frame*.

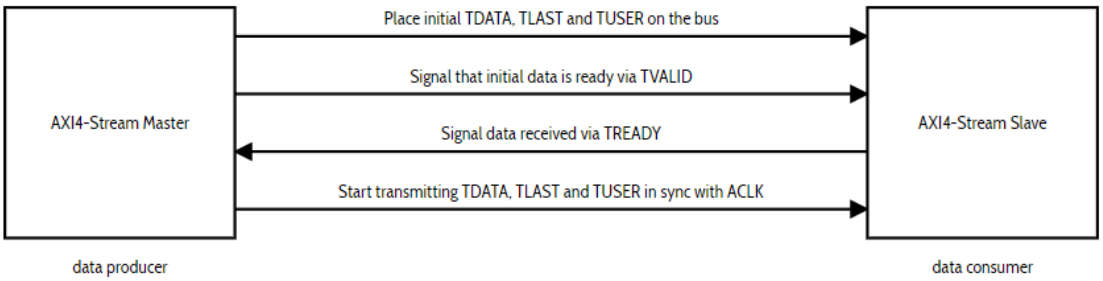


Figura 2.15: Interação entre mestre e escravo no protocolo *AXI4-stream*. Fonte: [32].

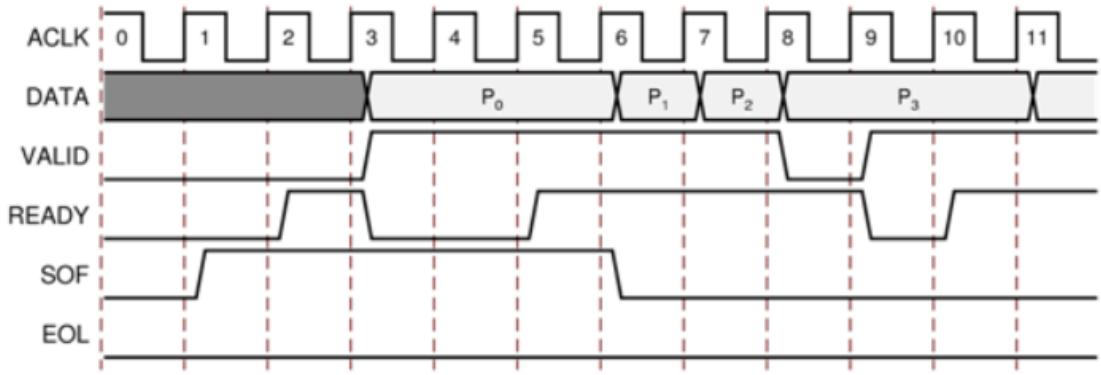


Figura 2.16: Exemplo para o começo de uma *frame* no protocolo *AXI4-stream*. Fonte: [32].

Estas duas *flags* são importantes para identificar a posição dos pixeis pois não existem sinais de *blanking* ou de sincronização [32].

2.8.2 Protocolo *AXI4-Lite*

Este protocolo AXI permite a configuração por *software* dos registos configuráveis, tanto para módulos IP da Xilinx, como para IPs desenvolvidos durante um projeto, em tempo de execução. Neste projeto é utilizado este protocolo para configurar os registo do do IP *Video Test Pattern Generator*, dos VDMAs e de alguns IPs implementados durante o projeto.

2.9 Módulos IP da Xilinx

Módulos de propriedade intelectual (IP) referem-se a funções lógicas pré-configuradas que podem ser utilizadas no *design* de um circuito lógico. Nesta secção, são abordados alguns módulos IP da biblioteca da Xilinx que são relevantes para este trabalho.

2.9.1 *Video Test Pattern Generator*

O *Video Test Pattern Generator* gera padrões de teste para a avaliação e *debugging* de sistemas de vídeo. Este módulo proporciona alguma variedade de padrões configuráveis. Estes assistem na avaliação da qualidade e performance de vídeo. O módulo apresenta as seguintes características:

- Formatos: RGB, YUV 444, YUV 422, YUV 420;
- Profundidade de cor de 8, 10, 12 e 16 bit;
- Apresenta barras de cores, padrões com degrau e velocidade ajustável, rampas temporais e espaciais, e retângulo móvel com tamanho e cor selecionáveis, sobre qualquer padrão de teste disponível;
- Resoluções espaciais desde 64 por 64, a 10328 por 7760 pixeis;
- Atinge resoluções até 4K a 60 *fps* em todas as famílias de dispositivos;
- Contêm interfaces *AXI4-stream*.

Na figura 2.17, é possível observar a saída *AXI4-stream* do módulo quando é configurado para gerar 2 pixeis por *clock*. Esta apresenta qual a componente de cor vai em cada parte de 10 bit do barramento.

59:50	49:40	39:30	29:20	19:10	9:0
		V0	Y1	U0	Y0

Figura 2.17: Saída do módulo para 10-bit YUV 4:2:2 com 2 pixeis por *clock*, V0 simboliza a crominância da cor vermelha do primeiro pixel, U0 a crominância da cor azul do primeiro pixel, Y1 e YO a luminância do segundo e do primeiro pixel respetivamente. Fonte: [7].

Na documentação da Xilinx [33] é possível encontrar mais informação relativa ao módulo, e inclusive informação de como gerar padrões.

2.9.2 *SMPTE UHD-SDI Transmitter Subsystem*

O *SMPTE UHD-SDI Transmitter Subsystem* implementa um transmissor para a família de normas SDI. Este módulo recebe vídeo a partir da interface *AXI4-stream* e envia uma *stream* de vídeo a ser transmitido segundo a interface SDI. Apresenta, também, as seguintes características:

- Suporta 2 pixeis por amostra;
- Suporta YUV 4:2:2;
- 10 bit por componente de cor;
- Apresenta uma interface *AXI4-stream*, que recebe a *frame* a enviar;

- Apresenta uma interface *AXI4-Lite*;
- Suporta diversas normas SDI, inclusive o 12G-SDI;
  - "*SMPTE ST 2082-1: 12G-SDI with data mapped by any ST 2082-x mapping at 11.88 Gbit/s and 11.88/1.001 Gbit/s*". [34]

Na figura 2.18, é possível observar a arquitetura do transmissor SDI e, na figura 2.19, a do recetor, que é outro módulo IP. Como este trabalho é realizado no lado do transmissor, não é apresentado muito detalhe do recetor.

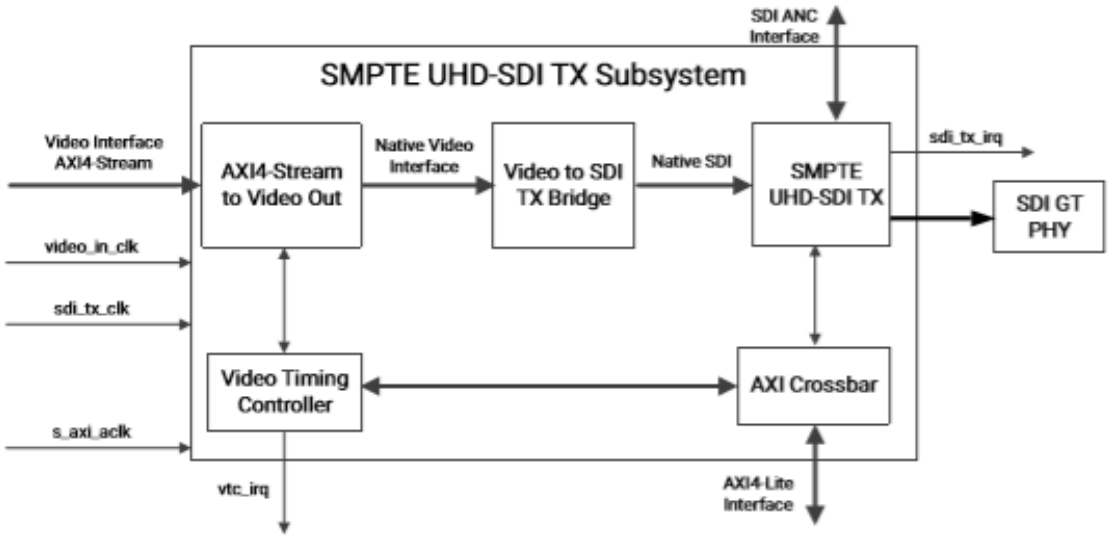


Figura 2.18: Arquitetura do transmissor. Fonte: [34].

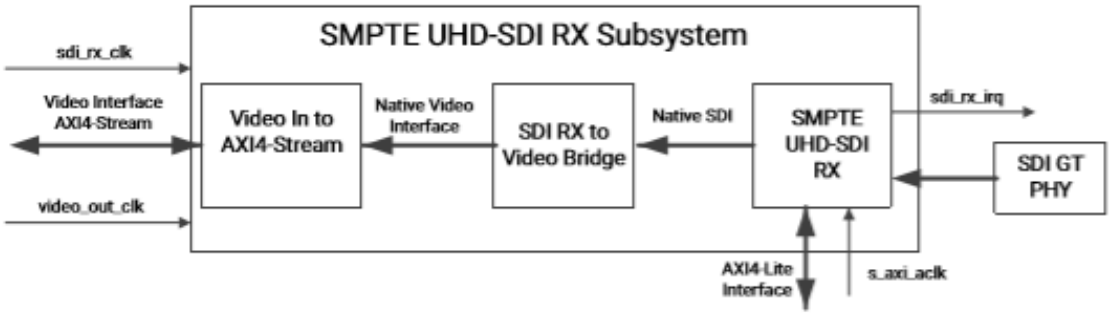


Figura 2.19: Arquitetura do recetor. Fonte: [35].

O transmissor SDI recebe uma *stream* pela interface *AXI4-Stream* e transforma os dados provenientes desta *stream* em conjunto com os provenientes da *SDI ANC Interface* em dados que podem ser enviados pela interface *SDI GT PHY*. Contém também uma interface *AXI4-Lite* que

permite controlar o módulo. Os sinais *video_in_clk*, *sdi_tx_clk*, e o *s_axi_aclk* representam o relógio do vídeo, relógio do módulo e relógio da interface *AXI4_Lite*, respetivamente.

Relativamente, aos módulos que o compõem, o módulo *AXI4-Stream to video Out* é responsável por transformar os dados da interface *AXI4-Stream* para uma para vídeo nativo. Por sua vez, o *video ro SDI TX bridge* transforma o vídeo nativo dados que podem ser enviados pela interface SDI e o *SMPTE UHD-SDI* recebe a informação do módulo anterior não multiplexada e gera uma *stream* de dados de 10 bit SDI multiplexados. O *video Timing Controller* controla o *timing* do vídeo e o *AXI Crossbar* é responsável por fazer o roteamento dos pedidos vindos de *AXI4-Lite*.

Este módulo apresenta diversos registos de configuração que estão divididos em dois grupos. Os registos do próprio módulo, que permitem configurar o tipo de interface, formato de vídeo e etc, e os registos do *Vídeo Timing Controller* que permitem configurar os *timings* do vídeo.

Na documentação da Xilinx [34], é possível encontrar mais informação relativa ao módulo, como por exemplo sinais necessários a ativar a cada ciclo de relógio.

2.9.3 *Video Broadcaster*

Este módulo da Xilinx também apresenta interfaces de dados *AXI4-stream*, uma de entrada e *n* (1 a 16) na saída. Como é necessário utilizar 4 interfaces SDI, este módulo seria configurado com 4 saídas para dividir a *stream*. Mas devido a um *bug* existente, foi necessário criar um IP que emula o comportamento deste e vai ser abordado mais à frente na implementação, mais propriamente no capítulo 7.

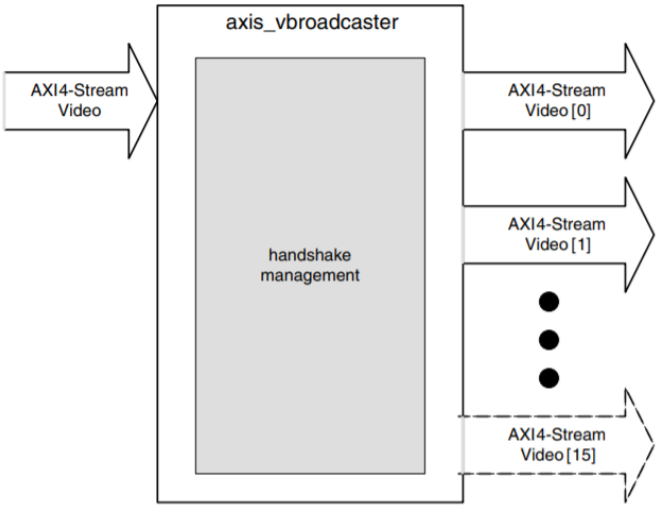


Figura 2.20: *Vídeo Broadcaster*. Fonte: [36].

2.9.4 *ILA (Integrated Logic Analyzer)*

Este IP da Xilinx é um módulo cujo principal objetivo não passa por contribuir para uma dada funcionalidade de um projeto. Este funciona com um analisador lógico que pode ser utilizado para

monitorizar os sinais internos de um projeto. Este módulo é síncrono com o restante *design*, e com isso, todas as restrições dos relógios do *design* são também aplicados aos componentes do *ILA*. Este módulo IP permite:

- Seleção do número de entradas de análise de sinal e o seu tamanho;
- Utilização de múltiplas provas que podem ter um único *trigger* em comum;
- Permite a análise de interfaces *AXI*, o que facilita o teste de módulos com este tipo de interfaces;

Na figura 2.21 está representada a interface deste IP. Os sinais do restante *design* na FPGA estão conectados às entradas prova e *clock*. Estes sinais são amostrados à velocidade do sinal de relógio que está ligado à entrada *clock* do módulo e os dados são guardados em blocos de RAM. Depois de o *design* ser carregado na FPGA, é utilizado o software de análise lógica do Vivado para provocar um evento que faça disparar a medição do *ILA* [37].

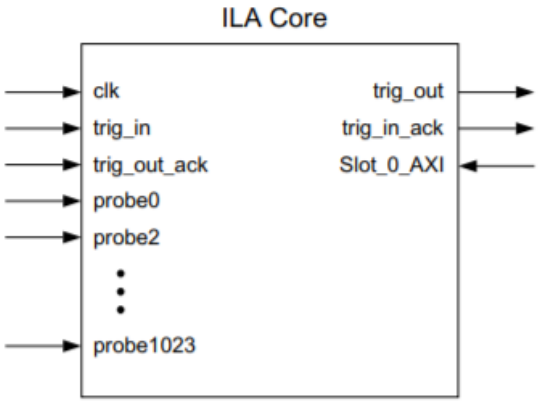


Figura 2.21: *Integrated Logic Analyzer*. Fonte: [37].

2.9.5 *VDMA (AXI Vídeo Direct Memory Access)*

Muitas aplicações de vídeo requerem *buffers* de *frames* para casos onde existam por exemplo, mudanças de *frame rate* ou de resolução. O *VDMA (AXI Vídeo Direct Memory Access)* é um IP da Xilinx que permite guardar em memória *frames* que chegam pela interface de vídeo *AXI-stream* [38] . Este módulo apresenta as seguintes características:

- Tamanho do barramento que envia para a RAM de 32, 64, 128, 256, 512 ou 1024 bit;
- Suporta larguras de dados AXI4-Stream com múltiplos de 8 até 1.024 bit;
- Consegue guardar até 32 *frames*.

Na figura 2.22 está representado o diagrama com os principais blocos constituintes do VDMA. Após os registos serem programados pela interface *AXI4-Lite*, o bloco *Control and Status* gera os comandos apropriados para o módulo *Data Mover*, que inicia a escrita ou leitura de dados vindo do módulo *Line Buffer* em memória através da interface *AXI4 Memory Map*. O módulo *Line buffer* implementa a funcionalidade de guardar uma dada quantidade de dados que chegam por *AXI4-stream* antes de enviar para a memória.

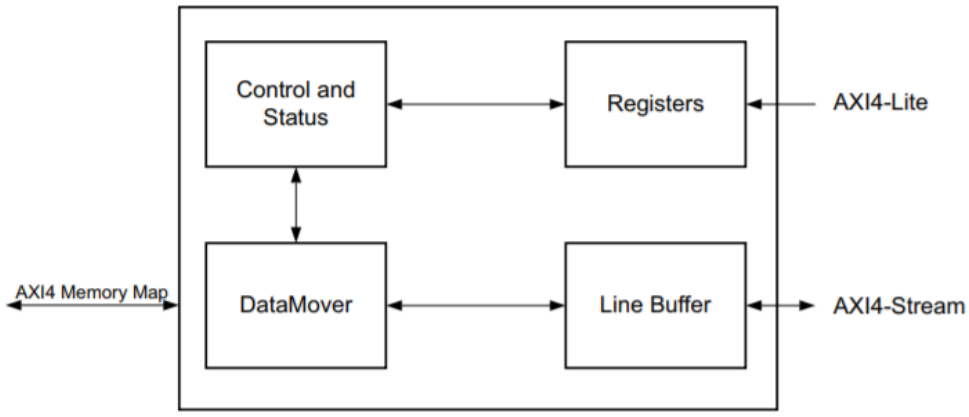


Figura 2.22: *Diagrama de blocos do VDMA*. Fonte: [38].

2.9.6 *AXI4-Stream FIFO*

A *AXI4-Stream FIFO* é um IP que permite acesso mapeado de memória para a interface *AXI4-stream* [39]. A principal aplicação deste IP consiste na leitura e escrita de pacotes de dados para ou de um dispositivo sem qualquer preocupação com o facto de estarmos perante uma interface *AXI4-stream*. Facilmente se consegue guardar dados que usam o protocolo *AXI4-stream* para serem transmitidos de uma forma transparente. Esta FIFO pode ser configurada para tamanhos de dados de 32, 64, 128, 256 ou 512 bit com uma profundidade de 512 a 128K posições.

2.10 Soluções concorrentes

A Omnitek apresenta soluções concorrentes às que foram implementadas durante este projeto. No site da Omniteck, podemos observar algumas dessas soluções [40], como é o caso dos Módulos IP e da *Ultra 4K Tool Box*.

2.10.1 *Ultra 4K Tool Box* da Omnitek

O *Ultra 4K Tool Box 4K* é um dispositivo gerador de padrões de vídeo utilizado para teste e análise. Este apresenta a desvantagem de apenas conseguir testar resoluções até 4096 x 2160 (4K) com 60 *fps*.

Esta é uma solução *plug & play*, pois permite a ligação direta dos cabos SDI às suas entradas e saídas e está pronta a ser utilizada, após a configuração do modo de funcionamento.

Este dispositivo é configurável para utilizar apenas um *link* de dados, dois ou quatro. Permite também escolher o método 2SI ou o *square division* para dividir a imagem. Pode ser utilizado para a análise das interfaces 12G SDI, 6G SDI, 3G SDI, HD SDI e SD SDI, e apresenta alguns padrões pré-configurados.

Na figura 2.23, é apresentado o conjunto de entradas e saídas da *Ultra 4K Tool Box*.

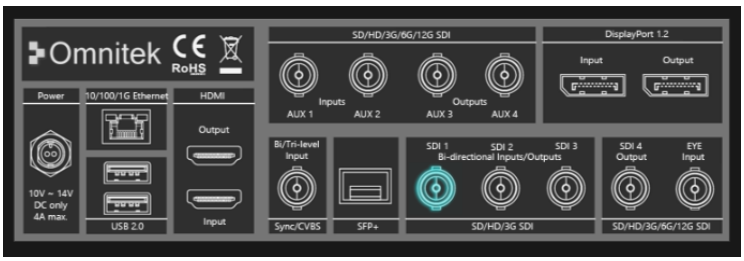


Figura 2.23: Entradas e saídas do *Ultra 4K Tool Box* da Omnitek. Fonte: [40]

2.10.2 Módulos IP que são comercializados pela Omnitek

Tal como a Xilinx, a Omnitek apresenta um conjunto de módulos nas suas bibliotecas com funcionalidades que podem ser utilizadas para gerar padrões de vídeo e distribuir imagens para diferentes tipos de interfaces SDI.

Um exemplo de módulos IP da Omnitek são os *SDI Rx & Tx Xilinx FPGA IP*. Este módulo permite a conversão de qualquer interface da SMPTE para uma interface AXI4-Stream, suportando até 12G SDI com apenas um *link*, 6G SDI com 2 *links* e 3G SDI com 4.

O SDI RX descodifica vídeo SDI para AXI4-Stream e o SDI TX, o contrário.

2.11 Trabalhos realizados no âmbito da transmissão em 8K

Nesta secção, são apresentados alguns trabalhos já realizados na transmissão de vídeo em UHD, mais propriamente em 8K, ou outros relevantes para a transmissão nesta resolução de vídeo.

"8K-TICO Codec for Miniaturized and Simplified UHDTV Production Systems", SMPTE 2017 Annual Technical Conference and Exhibition [6]

Segundo este trabalho, já foram realizadas experiências em 8K, utilizando 16 interfaces 3G SDI para atingir os 48 Gb/s (59.94 *fps*, 10 bit, 4:2:2), o que é muito dispendioso a nível do número de cabos a utilizar.

O trabalho a que se refere este artigo é relativo ao *"8K-TICO Codec"* [41]. Este foi utilizado pelos autores por necessitar de apenas um *link* de dados 12G SDI, pois afirmam ter vantagens relativamente ao uso de *multilink*. Este *codec* apresenta um rácio de compressão de 1/4 e consegue

transmitir uma vídeo 8K a 60 *fps*. O sinal com a resolução de 8K comprimido de 1/4 (aproximadamente a mesma informação que um de 4K) é enviado por uma interface 12G SDI ou por 4 3G SDI.

A utilização do *"8K-TICO Codec"*, permite atingir o 8K com qualidade e muito pouca latência, mas com "perda visual", o que não é desejável na indústria de vídeo profissional. Por outro lado, não existe degradação adicional por codificar e descodificar a imagem subsequentemente. Na tabela 2.4, é possível observar as especificações do *"8K-TICO Codec"*.

Tabela 2.4: Especificação da utilização do *"8K-TICO Codec"*. Fonte: [6].

	Specification	Remarks
Input to Encoder	8K: 4 × 12G-SDI 4K: 1 × 12G-SDI or 4 × 3G-SDI	7680 × 4320 3840 × 2160
Output from Decoder	8K: 4 × 12G-SDI 4K: 1 × 12G-SDI or 4 × 3G-SDI HD: 1 × 3G-SDI (1080p)	 Down-converted from 8K Down-converted from 4K
TICO code stream	8K: 1 × 12G-SDI or 4 × 3G-SDI 4K: 1 × 3G-SDI	1/4 compression 1/4 compression
Chroma sample/bit depth	4:2:2 / 10 bits	
Video data mapping	8K: 2SI 4K: 2SI or SQD	
Size of a unit	483(W) × 196(D) × 44(H) mm	1 RU
Size of a module	189 × 160 mm	
Power consumption	Less than 80 W	100-240 VAC

UHD in a Hybrid SDI/IP World [2]

Este estudo afirma que a utilização de redes IP em exclusivo pode ser uma boa aproximação, mas um sistema onde a comutação crítica de tempo (em tempo real) usa SDI e, a não crítica usa IP (transferência de ficheiros), é um modelo bem estabelecido.

Para além disto, aborda algumas normas da SMPTE, confirmando o que foi afirmado neste capítulo relativamente ao número de sub-imagens utilizadas para 8K (16 imagens).

Design and Implementation of 8K UHD Encapsulation Method for Efficient Transmission and Reception based on MMT [4]

Este trabalho consiste na implementação de um método de encapsulamento 8K baseado em MTT (*MPEG media transport*). O MMT é um protocolo de transmissão baseado em IP.

O estudo afirma que através da tecnologia atual é difícil proporcionar 8K em redes *broadcast* derivado dos elevados *bitrates* (48 Gbit/s). Portanto, é utilizado o *codec* SHVC (*Scalable High Efficiency video Coding*).

Esta metodologia divide a imagem original em 4 sub-imagens cada uma com 3 camadas (figura 2.24), existindo portanto 12 (3 ×4) *streams* de vídeo. As camadas 0 e 1, informação auxiliar e o áudio são encapsulados em pacotes MMT e, posteriormente, transmitidos após passar por um

multiplexador. A camada 2 é transportada por uma rede de comunicação. Com este método os utilizadores podem ver diferentes resoluções (8K, 4K, *Full*-HD) dependendo do ambiente em que se encontram. Neste artigo, os resultados foram os esperados, mas apenas foi transmitida informação a 30 *fps*.

No canto superior esquerdo da figura 2.24, é possível observar as 12 *streams* de vídeo mencionadas no texto anterior, divididas em 3 camadas. As camadas 0 e 1 vão para o *MMT Generator* em conjunto com o áudio e a informação auxiliar. A camada 2 vai para o *HTTP Streaming Server*. O lado direito da figura demonstra a utilização de diferentes resoluções dependendo do ambiente em que se encontra o utilizador.

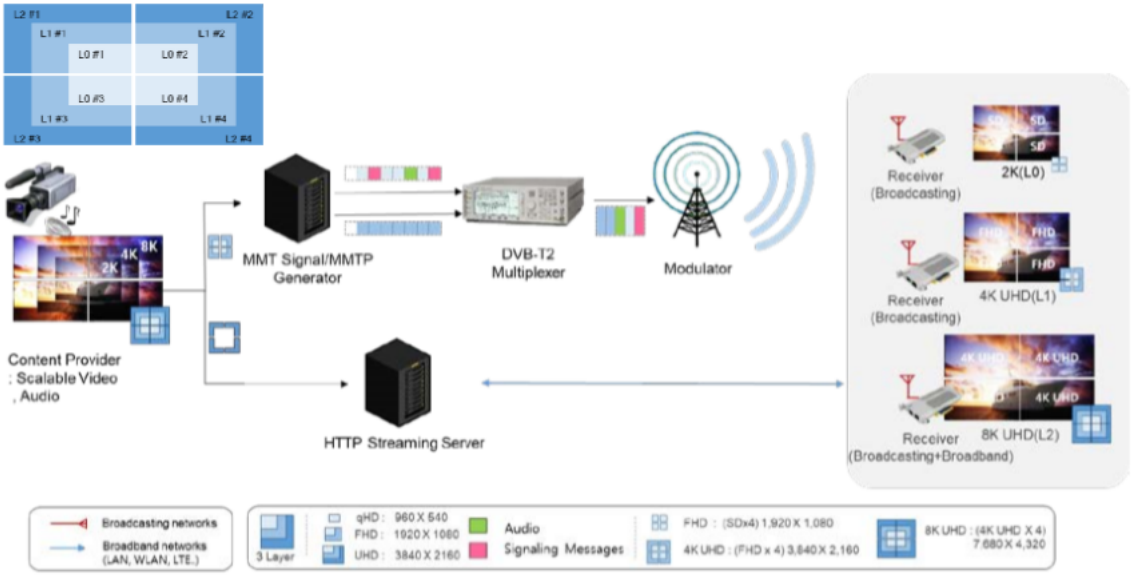


Figura 2.24: Arquitetura para o *broadcast* de 8K. Fonte: [4].

Para sumarizar, este capítulo apresenta as tecnologias e fundamentos, normas, módulos IP que serão utilizados, soluções concorrentes e trabalhos que já foram realizados no âmbito do 8K em que se baseia este projeto. O capítulo 3 apresenta os estudos realizados para constituir uma arquitetura e um plano de teste antes da implementação do sistema.

Capítulo 3

Estudo do sistema desenvolvido

Este capítulo começa por apresentar uma *overview* do projeto desenvolvido descrito nesta dissertação. Também são apresentados alguns estudos que foram realizados antes da implementação do projeto, como é o caso da análise de requisitos, estudo da escalabilidade, desenvolvimento da arquitetura, plano de validação do sistema e características da FPGA a utilizar.

3.1 *Overview* do sistema

O sistema desenvolvido neste projeto tem como objetivo gerar padrões de vídeo e enviá-los para uma interface de vídeo com 4 *links* SDI. A primeira parte do projeto, e que está presente neste capítulo, passou pela análise de requisitos do sistema, desenho de uma arquitetura e desenvolvimento de um plano de validação. Só depois destes pontos estarem bem definidos é que foi iniciada a implementação.

Durante a definição da arquitetura, o sistema foi dividido em 4 subsistemas principais sendo que o primeiro é apresentado no capítulo 4, o segundo no 5, o terceiro no 6 e o quarto no 7. Os capítulos não seguem a mesma ordem segundo o qual o sistema foi desenvolvido pois, neste relatório, para melhor entendimento do leitor, os capítulos foram escritos consoante o fluxo dos dados. Para além desta opção, estes 4 capítulos apresentam a implementação, teste e resultados de cada um dos subsistemas.

Para finalizar o desenvolvimento do projeto, o capítulo 8 apresenta os diversos subsistemas como um sistema completo e capaz de enviar os padrões de vídeo para os módulos responsáveis pelo envio para a interface SDI. Ou seja, apresenta a integração dos subsistemas desenvolvidos com os módulos responsáveis pelo envio para os *links* SDI.

3.2 Análise de requisitos

Visto que o sistema tem de obedecer a certas regras e requisitos de maneira a contribuir para outras plataformas em desenvolvimento na empresa, realizou-se uma análise de requisitos que o

sistema tem de cumprir, antes de se definir uma arquitetura. Após pesquisa e discussão do que era necessário e exequível com a MOG, os principais requisitos para o sistema são:

- Suportar formato de vídeo YUV 422 10-bit;
- Suportar a resolução 8K (7680 x 4320) e a resolução 4K (3840 x 2160);
- Suportar *frame rates* inteiros e fracionários;
- Suportar os *frame rates* 24, 25, 30, 29.97, 48, 50, 60 e 59.94 *fps*;
- Suportar vídeo progressivo;
- Apresentar diversos padrões de vídeo com movimento, um objeto intermitente e o *timecode* a sobrepor o padrão;
- Ser capaz de fornecer dados que consigam ser enviados por uma interface de vídeo com 4 *links* de dados para, posteriormente enviar para a interface SDI.

3.3 Escalabilidade

Como já foi referido no capítulo 2, as resoluções de vídeo têm vindo a evoluir para resoluções com maior número de pixeis e com diferentes *frame rates*. Devido a este facto, ao desenvolver a arquitetura que é apresentada na próxima secção, o sistema teve se der idealizado para:

- Suportar diversas resoluções e *frame rates*;
- Ser facilmente alterável para adicionar novas resoluções e *frame rates*, que para já não foram necessários considerar.

De tal forma, a arquitetura foi desenhada para não trabalhar só com uma resolução ou *frame rate*, mas sim com vários. Foi também pensada de maneira a facilitar a adição de mais tipos de resolução ou *frame rates*. Para tal estes são configuradas por *software* e para adicionar mais, é só adicionar as novas resoluções e *frame rates* nos diferentes módulos.

3.4 Arquitetura do sistema

Na fase de desenvolvimento de uma arquitetura, a abordagem escolhida consiste em dividir o projeto em 4 subsistemas principais, que interagem entre si. Existe ainda mais um subsistema secundário para configurar estes 4 principais e o subsistema composto pelos módulos que enviam para os *links* SDI. Este dois últimos subsistemas só são abordados no capítulo que discute o sistema completo.

Após o padrão ser gerado no subsistema representado pelo bloco *Video Pattern Generator* na figura 3.1, este passa por dois subsistemas que lhe vão adicionar dados complementares (um

deles acrescenta um retângulo que intermitente e outro um *timecode* sobreposto sobre a *frame*). O retângulo intermitente tem o objetivo de permitir verificar o sincronismo entre o vídeo e o áudio (desenvolvido noutro projeto a decorrer em paralelo na MOG) e o *timecode* tem o objetivo de permitir a comparação dos dados presentes nos metadados com o próprio *timecode* que vai na imagem e confirmar que o vídeo está ser enviado no *timing* correto. Posteriormente o vídeo chega ao subsistema denominado por *2SI* que tem a função de dividir a *stream* de vídeo em 4 *streams* com 1/4 da resolução da original. O método escolhido de entre os apresentados na secção 2.5 foi o 2SI, tal como o nome do seu subsistema indica.

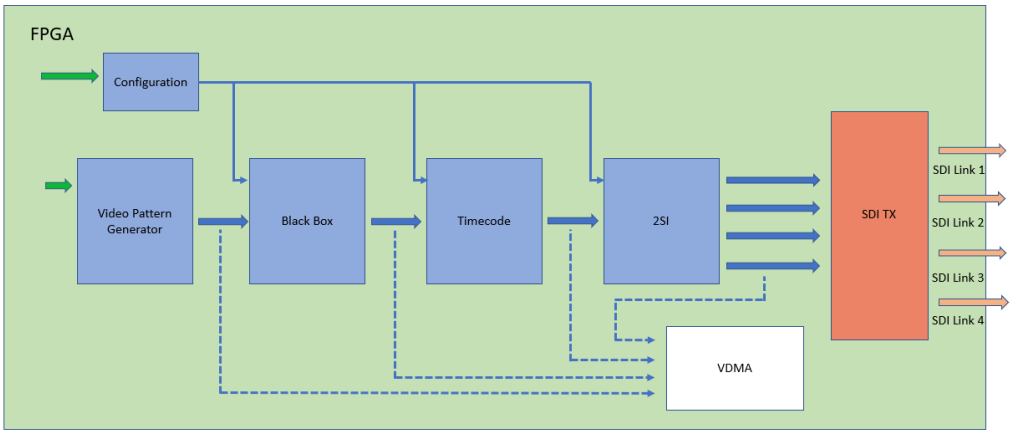


Figura 3.1: Arquitetura geral.

O subsistema representado a laranja na figura 3.1, não tem de ser desenvolvido no projeto, apenas têm de ser configurados os IPs responsáveis por receber os dados que vêm do subsistema *2SI* por *AXI4-stream* e transforma-los em dados que possam ser enviados por *links* SDI. Isto é importante para efeito de teste em condições reais do sistema desenvolvido neste projeto. No entanto ainda não se encontra a funcionar devidamente. Desde o início do projeto já foi tido em conta que na fase final do mesmo, este subsistema poderia não se encontrar a funcionar devido à falta de informação necessária para o configurar corretamente de forma a trabalhar com altas cadências de dados. Isto provoca algumas implicação na fase de teste do sistema pois impossibilita o teste em condições reais de utilização.

O bloco a branco denominado *VDMA* apresenta setas a tracejado que vão na sua direção, pois este bloco nem sempre esteve no mesmo sítio do *design*. A utilização deste bloco tem o objetivo de guardar *frames* em memória, e sempre que durante o projeto se pretendeu fazer um teste de um dado subsistema, este era colocado à saída do módulo a testar (para módulos com interfaces AXI), gerado novo *bitstream*, e realizados novos testes.

Quanto ao bloco *Configuration*, este permite receber a configuração da resolução e *frame rate* através de uma interface AXI4-lite (setas verdes representam este tipo de interfaces) que depois são distribuídas pelos restantes módulos que o necessitam. O módulo IP *Video Test Pattern Generator* presente no módulo *Video Pattern Generator* também é programável por este tipo de interface.

Na figura 3.2, é possível observar um esboço de *frame* da *stream* que se pretende obter antes de se dividir no subsistema *2SI*. A parte azul é gerada pelo subsistema *Video Pattern Generator*, a parte a branco representa o *timecode* gerado pelo subsistema *Timecode* e a parte preta representa a componente gerada pelo subsistema *Black Box*.

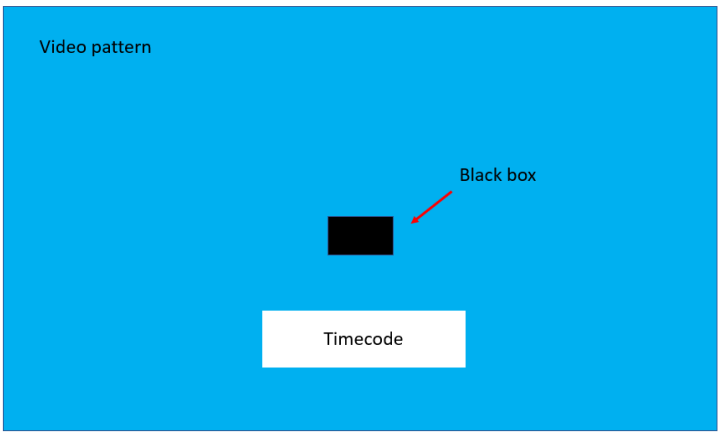


Figura 3.2: Esboço de uma *frame* antes de ser dividida no subsistema *2SI*.

Devido ao facto do subsistema *SDI TX* não estar a funcionar, durante o desenvolvimento do projeto foi pensada uma arquitetura que permite testar os subsistemas antes deste subsistema. Para tal utilizou-se a arquitetura representada na figura 3.3, que é composta pelo subsistema secundário de configuração, os 4 subsistemas principais (*Video Pattern Generator*, *Black Box*, *Timecode*, *2SI*) e 4 VDMAs para guardar em memória dados que seriam enviados para os 4 *links* SDI. Os VDMAs permitem guardar dados das *frames* que recebem, o que possibilita a análise do funcionamento do sistema enquanto o subsistema responsável pelo envio para o SDI não se encontra a funcionar e não é possível observar o sinal à saída dos links SDI.

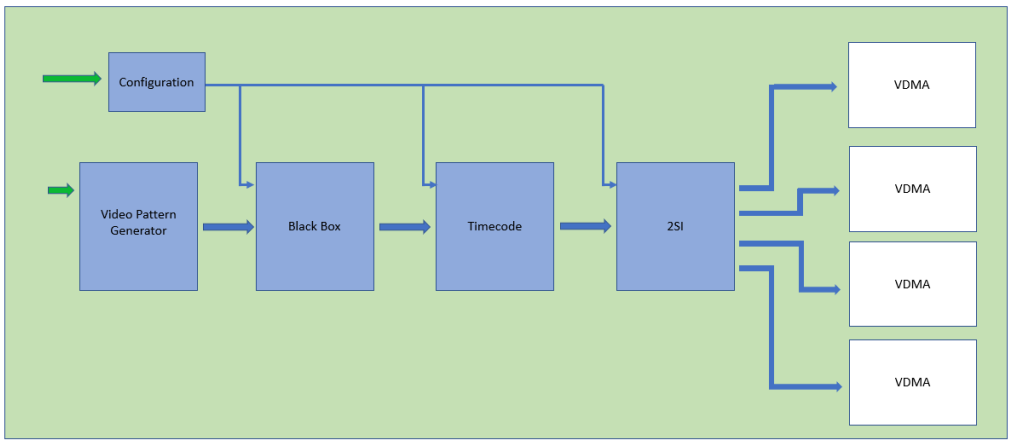


Figura 3.3: Arquitetura final do sistema com 4 *VDMA*.

3.5 Plano de validação

Após desenhar uma proposta de arquitetura foi realizado um plano de validação para validar este sistema. Este plano pode ser dividido em duas fases, em que a primeira é a validação através de um *testbench* em *System Verilog* e a segunda é a validação após a implementação na FPGA.

3.5.1 Simulação

O *testbench* desenvolvido para testar o sistema em fase de simulação foi a primeira etapa do projeto (figura 3.4). Este foi pensado para poder ser utilizado sem serem necessárias grandes alterações ao longo deste trabalho. O *testbench* tem o objetivo de programar os subsistemas *Configuration* e o *Video Pattern Generator* através da interface *AXI VIP*, e seguidamente analisar as saídas dos diferentes módulos.

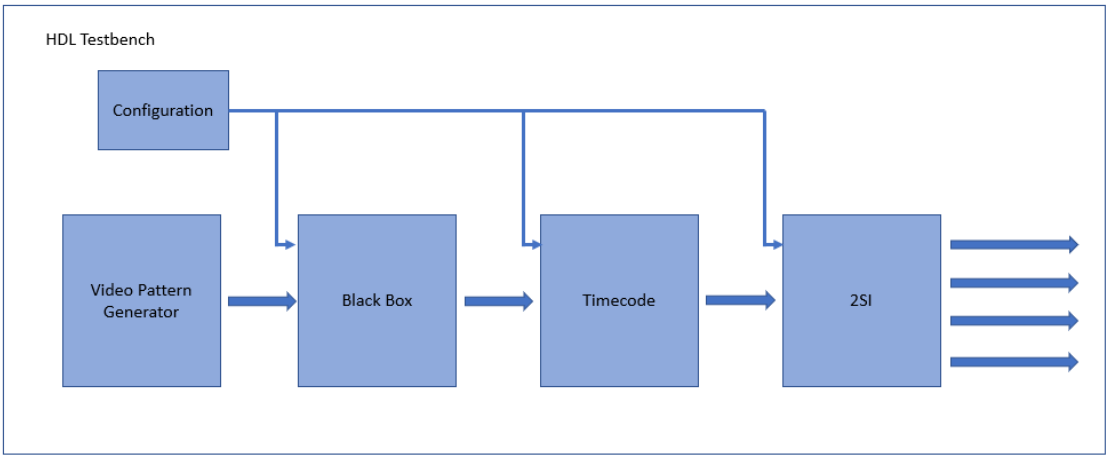


Figura 3.4: Sistema em fase de simulação.

Este *testbench* tem uma particularidade que lhe permite ser reutilizável e validar se o sistema está a funcionar corretamente sem ser sempre necessário olhar para as formas de ondas. O *testbench* guarda num ficheiro binário os dados referentes à *stream* de vídeo que chegam por uma interface *AXI4-stream*. Ou seja, é capaz de interpretar o protocolo *AXI4-stream* e descodificar as *frames* que compõem o vídeo. Isto permite (através da análise do ficheiro binário com um programa que o veja como um vídeo no formato YUV 422, o YUV VIEWER [42] por exemplo) observar a *stream* em qualquer ponto do sistema.

Para alterar o ponto onde se deseja visualizar a *stream* apenas é necessário alterar a interface *AXI-stream* que se pretende observar, redefinindo os sinais da mesma, como saídas do sistema. No *testbench* não é necessária nenhuma alteração. Este guarda num ficheiro os dados que recebe pela interface *AXI4-stream* configurada no *desing*.

Por exemplo, se se pretender observar o vídeo entre o subsistema *Black Box* e o subsistema *Timecode* da figura 3.4, no *design* ligam-se os sinais da interface que os conecta às saídas para

serem adquiridas pelo *testbench*. Neste caso, consegue-se visualizar o vídeo já com o retângulo intermitente, mas sem tudo o que está para a frente, o que permite:

- Observar erros no que está para trás deste módulo;
- Testar parcialmente o sistema à medida que vai sendo construído.

3.5.2 Sistema implementado em FPGA

Para testar o sistema após a sua implementação em FPGA foi também realizado um plano de testes semelhante ao plano tomado na simulação, e está demonstrado na figura 3.5. Este plano de teste pós-implementação consiste em utilizar os VDMAs, que permitem guardar 32 *frames* em memória. Realizando um *dump* de memória é possível observar as *frames*. Durante as diversas fases de implementação é possível observar as *frames* à saída de diferentes subsistemas posicionando os VDMAs em diferentes partes do projeto. Os VDMAs estão conectados aos diversos módulos por interfaces *AXI-stream*.

A aplicação de *software* desenvolvida que corre no CPU (*Micro Blaze* [43]) foi elaborada na linguagem C. Esta aplicação começa por realizar a configuração dos diversos módulos antes de iniciar a geração de padrões. Quando começa a gerar a *stream* de vídeo guarda os dados nos VDMAs.

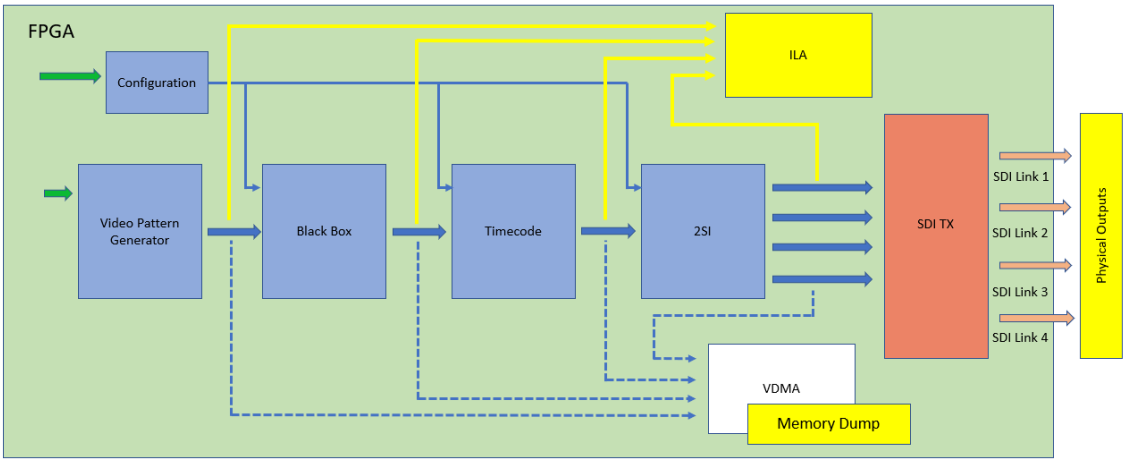


Figura 3.5: Teste utilizados após implementação em FPGA.

Durante o desenvolvimento do sistema, também é utilizado um *ILA (Integrated Logic Analyzer)* que permite observar as formas de ondas desejadas enquanto o sistema está a correr.

O último teste planeado consiste na aquisição do sinal nas saídas da FPGA e observa-las num ecrã. A placa escolhida não contém saídas nativas SDI, no entanto podem ser utilizadas saídas QSFP, em que se conecta um *QSFP28 Connectors Module*, que apresenta um *transceiver* LMH1297[44], externo à FPGA, permitindo selecionar a direção de cada canal SDI. Um QSFP28 pode ser utilizado com a configuração de um *transceiver* com 4 canais, um para cada *link* SDI.

No entanto, este teste não pode ser realizado enquanto o subsistema que se encarrega de enviar o sinal para os *links* SDI não estiver a funcionar; e como no final do tempo alocado ao projeto o subsistema ainda não se encontrava funcional não foi realizado para já.

3.6 Placa utilizada

A placa escolhida para a realização deste projeto foi a VCU 1525 e o seu aspeto físico pode ser observado na figura 3.6. Quanto à sua organização interna, esta está descrita no diagrama de blocos da figura 3.7 e apresenta uma FPGA *Virtex UltraScale+ XCVU9P-L2FSGD2104E* [45]. Para além da FPGA, neste projeto são utilizados os seguintes recursos da placa:

- QSFP (*Quad Small Form-factor Pluggable*) para dar *output* do sinal SDI, visto que a placa não tem interfaces nativas SDI;
- Sinais de relógio. O SDI necessita de uma frequência de relógio específica de 148.5 MHz para as frequências inteiras e 148.35 para as fracionárias. A placa tem um SI 570 (*clock* programável por I2C)
- Blocos de memória C0 a C3 onde os VDMAs guardam os dados. Estes, são memórias DDR4 RDIMM de 16 GB e suportam 2400 MT/s (*Mega Transfers per second*).

Figura 3.6: VCU 1525. Fonte: [45].

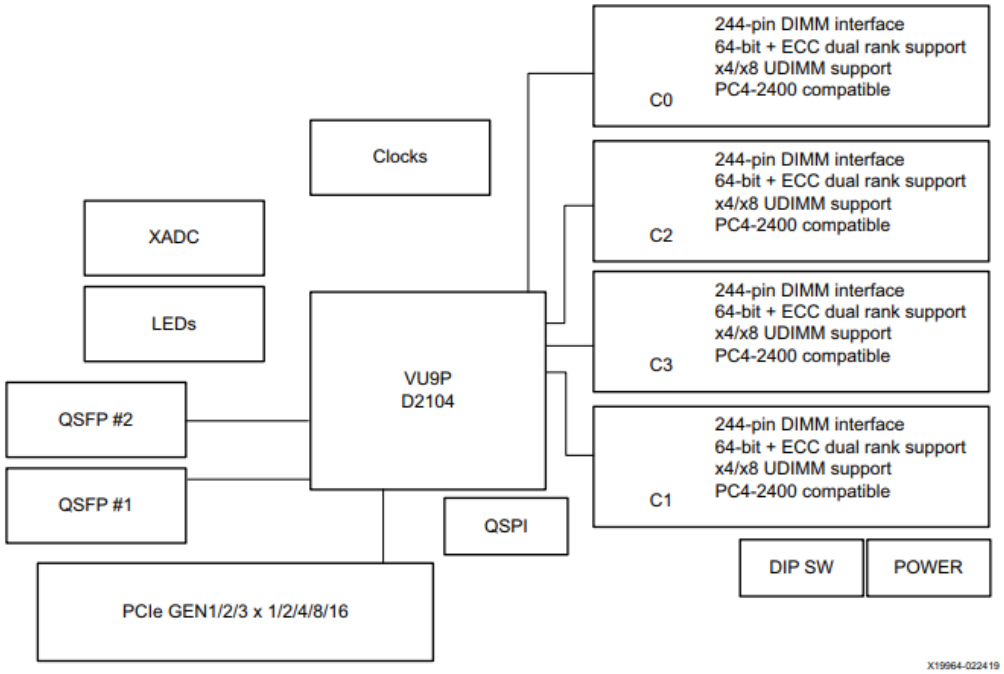


Figura 3.7: Diagrama de blocos da VCU 1525. Fonte: [46].

3.7 Ligação da saída QSFP ao *tranceiver* externo

Como já foi mencionado no plano de testes, uma das saídas QSFP da FPGA encontra-se ligada a um *QSFP28 Connectors Module* que permite selecionar a direção de cada um dos 4 canais SDI. Neste projeto todos os canais são selecionados como TX, pois são necessários 4 *links* de dados SDI TX. Estes *links* SDI podem ser ligados ao TX com o objetivo de reconstruir a imagem, mas se quisermos olhar só para um *link* com 1/4 da resolução da imagem original podemos ligar diretamente a um monitor com este tipo de entrada. Nesta FPGA as duas saídas QSFP apresentam *trancievers* GTY com 4 canais e, permitem transformar os dados de série para paralelo e vice versa, sendo que do lado do TX os dados são passados de paralelo para série.

Capítulo 4

Subsistema que configura o gerador de padrões de vídeo

Neste capítulo é apresentada a primeira etapa realizada durante a implementação do sistema (subsistema 1 mencionado na arquitetura). A primeira parte desta etapa consiste em configurar o módulo *Video Test Pattern Generator* da Xilinx, de maneira a obter a cadência de dados desejada para, posteriormente, enviar pela interface SDI. O Sistema foi idealizado de forma a suportar 8K 60 *fps* por uma interface SDI 12G *quad-link*, ou seja, com uma cadência 48 Gbit/s.

4.1 Implementação

A implementação deste módulo pode ser dividida em duas partes, a parte de configuração do IP *Video Test Pattern Generator* e a parte de condicionamento de sinal proveniente deste módulo.

4.1.1 Configuração do IP *Video Test Pattern Generator*

Para a resolução 8K, a 60 *fps*, e formato YUV 422 10-bit existem 39813120000 bit/s (valor obtido a partir da fórmula 4.1), ou seja, aproximadamente 40 Gbit/s de dados relativos à imagem que necessitam de ser enviados. O módulo *Video Test Pattern Generator* possui a limitação de apresentar como valor máximo para a sua frequência 300 MHz e, portanto, os restantes módulos responsáveis pelo envio para a interface SDI vão trabalhar a esta frequência.

$$Cadência\,(bit/s) = bit\ por\ pixel \times frame\ rate \times resolução\ horizontal \times resolução\ vertical \quad (4.1)$$

Este módulo pode ser configurado com 1,2,4 ou 8 pixeis por ciclo de relógio. Dada a frequência de relógio de 300 MHz, para uma resolução 8K 60 *fps*, no formato YUV 422-10 *bit*, variando o número de pixeis por ciclo de relógio, obtêm-se no máximo, as quantidades de bit/s demonstradas na tabela 4.1 e calculados pela fórmula 4.2.

$$Cadência\,do\,Video\,Test\,Pattern\,Generator(bit/s) = bit\,por\,pixel \times pixeis\,por\,clock \times \text{frequência} \tag{4.2}$$

Tabela 4.1: Cadência do *Video Test Pattern Generator* dependendo da configuração de pixeis por ciclo de relógio para a resolução 8K 60 *fps*.

Amostras por ciclo de relógio	**Bit/s**	**Gbit/s**
1	6000000000	6
2	12000000000	12
4	24000000000	24
8	48000000000	48

Observando a tabela 4.1, verifica-se que a única configuração que gera uma cadência de dados suficiente é a de 8 pixeis por ciclo de relógio, pois é necessário enviar dados a aproximadamente 40 Gbit/s e esta é a única configuração que fornece cadência de dados máxima superior a 40 Gbit/s (fornece a 48 Gbit/s).

A utilização da configuração de 8 pixeis por ciclo de relógio também facilita a implementação dos módulos responsáveis pela aplicação do método 2SI, que vai ser explicado mais à frente no capítulo 7. Para uma *stream* 4K é necessário configurar o gerador de padrões de vídeo com 2 pixeis por ciclo de relógio para ter dados suficientes. Como uma *stream* 8K apresenta 4 vezes mais pixeis por *frame* do que uma 4K, é normal que com a mesma frequência de *clock* e mesmo *frame rate* sejam necessários 4 vezes mais pixeis a sair do *Vídeo Test Pattern Generator* [34].

Relativamente ao número máximo de colunas e de linhas foram escolhidos os correspondentes ao 8K (7680 para as colunas e 4320 para as linhas), pois é a resolução máxima que se pretende atingir, por enquanto. Sendo assim, os valores relativos ao 8K chegam para qualquer outra resolução que se pretende para este projeto e ao configurar o número máximo de linhas e de colunas com valores mais altos provocaria um desperdício de recursos desnecessário. O quadrado com movimento está ativo e todos os tipos de padrões estão ativos, pois é necessário um sistema com movimento e com vários padrões diferentes. Na tabela 4.2, são demonstradas as configurações escolhidas e mencionadas neste capítulo.

Este projeto tem como objetivo a implementação em FPGA de um gerador de padrões de vídeo UHD para a interface SDI no formato YUV 422 10-bit. No entanto, durante uma primeira simulação do *Video Test Pattern Generator*, com o objetivo de visualizar as suas saídas, foi observado que ao configurá-lo com 10 bit por componente de cor, na realidade não utiliza os 10 bit para representar uma componente de cor. Os 8 mais significativos mudam consoante a componente de cor correspondente e os dois menos significativos estão sempre a 0. Portanto, optou-se por uma configuração com 8 bit e adicionar os tais zeros para formar os 10 bit mais para a frente no sistema. Utilizar este módulo com 8 bit e seguir sempre com 8 bit até ser necessário utilizar 10 traz as seguintes vantagens:

- Poupança de recursos da FPGA, pois os barramentos são menores;
- Ao guardar o vídeo gerado, tanto em simulação como em execução em FPGA, o facto de ter sido guardado com 8 bit faz com que seja mais fácil e intuitivo tratar do ficheiro gerado.

Tabela 4.2: Configurações do módulo *Video Test Pattern Generator.*

Configurações do módulo *Video Test Pattern Generator*	
Pixeis por ciclo de relógio	8
bit por pixel	8
Máximo número de colunas	7680
Máximo número de linhas	4320
Tipos de padrão	Todos
Quadrado com movimento	Ativo

4.1.2 Condicionamento do sinal vindo do *Vídeo Test Pattern Generator*

Após a configuração do *Vídeo Test Pattern Generator* com os dados da tabela 4.2, o barramento de dados da interface *AXI4-stream* à saída deste módulo é automaticamente gerado com o tamanho de 192 bit. Destes 192 bit, apenas os 128 menos significativos contêm a informação que é necessária transmitir. Os 64 bit mais significativos contêm informação igual à correspondente aos bit [127:120], que é nada mais nada menos que a componente vermelha da crominância de um dos 8 pixeis transportados no barramento de dados. No diagrama de blocos da figura 4.1, está representado o módulo que gera os padrões de vídeo mais à esquerda, e o que condiciona o sinal mais à direita (*YUV filter*). O sinal *m0_axis* representa a interface *AXI4-stream* que liga estes dois módulos e apresenta um barramento de dados com 192 bit. O sinal chega ao módulo *YUV filter* com um barramento de dados de 192 bit e à saída, o sinal m1_axis apresenta um barramento de dados 128 bit. Os restantes sinais que entram pela interface *AXI4-stream m0_axis* do módulo *YUV filter* são diretamente atribuídos à interface mestre do módulo e entregues ao próximo subsistema.

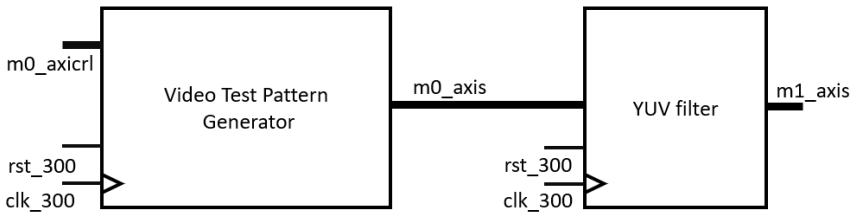


Figura 4.1: Diagrama de blocos do subsistema responsável por gerar os padrões de vídeo e de condicionamento do seu sinal.

4.2 Teste

Como já foi referido no capítulo 3, para testar o subsistema antes de implementar na FPGA, foram realizadas simulações comportamentais do circuito. Nesta fase do projecto, o *testbench* adquire as *frames* à saída do *YUV filter*, que são guardadas num ficheiro binário. Para visualizar este ficheiro foi utilizada a ferramenta YUV VIWER [42], que permite observar o ficheiro binário como uma *stream* de vídeo no formato YUV 422 8-bit.

Após os resultados serem os esperados, o sistema foi implementado na FPGA e novamente testado. Para testar o sistema foi adicionado ao projeto um VDMA conectado à saída do módulo *YUV filter*. O VDMA permite adquirir 32 *frames* e, para visualizar o vídeo foi realizado um *dump* dos valores guardados em memória relativos às 32 primeiras *frames* da *stream* e analisados novamente no YUV VIEWER.

Foram realizados os seguintes testes com o *Vídeo Test Pattern Generator* configurado por *software* para:

1. Resoluções padrão (8K, 4K ...)
2. Resoluções fora do padrão com o número de colunas múltiplo de 8 (4000 x 4000, por exemplo)
3. Resoluções fora do padrão com o número de colunas não múltiplo de 8 (3999 x 3999, por exemplo)

É de notar que este foi o primeiro subsistema em que foi aplicada a metodologia de teste apresentada no capítulo 3. Recapitulando, tanto nas simulações comportamentais como no teste após implementação na FPGA, foram colocados mecanismos à saída do subsistema que permitem adquirir *frames* do vídeo no protocolo *AXI4-stream* e analisar o correto funcionamento do sistema.

4.3 Resultados

Na figura 4.2 é possível observar uma *frame* 8K com o padrão *"Taratan Bars"* retirada do VDMA. Nesta fase do projeto, este era o resultado esperado para o subsistema, ou seja,uma *stream* de um padrão de vídeo à sua saída, pronta para enviar para o próximo subsistema. Esta *frame* corresponde ao teste 1 mencionado acima.

Basicamente, o principal objetivo dos *testbench*/VDMAs é este mesmo. Em caso de existir um problema no sistema será mais fácil descobri-lo através de uma imagem/vídeo em vez de olhar para as formas de ondas. Muitos dos problemas conseguem ser descobertos desta forma, como por exemplo se a ordem das componentes dos pixeis estão trocadas. Caso não seja possível identificar o problema olhando para a imagem/vídeo, não resta outra solução a não ser analisar as ondas.

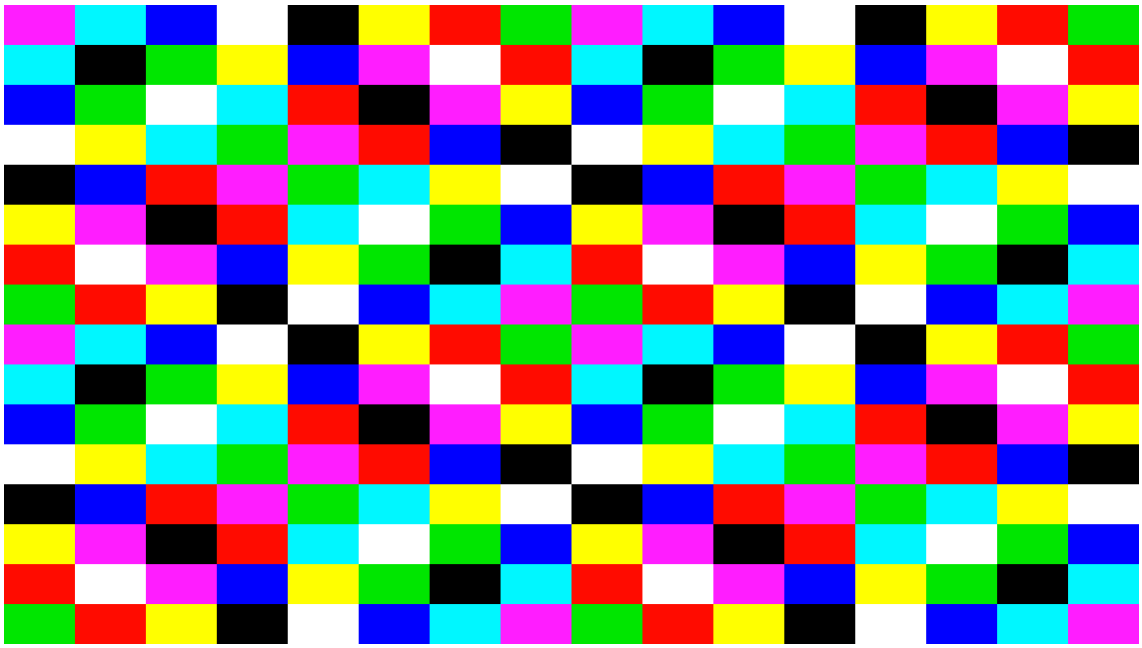


Figura 4.2: *Frame* 8K retirada do VDMA com o padrão "Taratan Bars"para o teste do subsistema *Vídeo Pattern Generator*.

Relativamente aos testes 2 e 3, estes serviram para explorar mais profundamente o *Vídeo Test Pattern Generator* e permitiram concluir que este módulo só funciona corretamente se a resolução que é configurada por *software* apresentar o número de colunas múltiplo do número de pixeis por ciclo de relógio, que neste caso é 8. Este módulo implica que o sistema final só funcione para resoluções com o número de colunas múltiplo de 8, o que não é muito limitativo, porque apenas estamos interessados nas resoluções padrão e estas apresentam todas números de colunas múltiplos de 8.

Após os resultados para esta fase do projeto serem os esperados, foi desenvolvido o subsistema *"2SI"*, mas no próximo capítulo vai ser abordado o "Black Box"pois é o próximo subsistema na cadeia de dados.

Capítulo 5

Subsistema que gera o objeto intermitente

Após a filtragem do sinal que sai do gerador de padrões de vídeo, vão ser adicionados elementos gráficos à imagem. O primeiro a ser adicionado, e que vai ser descrito neste capítulo é um retângulo intermitente. Este retângulo vai ser gerado pelo subsistema *Black Box* e tem como objetivo permitir a análise do sincronismo entre o vídeo e o áudio a transmitir. Quando estiver a ser enviado vídeo e áudio para a interface SDI, se o retângulo for configurado, por exemplo para piscar de um em um segundo, e o áudio também estiver ativo de um em um segundo ao mesmo tempo que o vídeo, é possível perceber se ambos estão sincronizados.

O subsistema apresentado neste capítulo apenas apresenta um módulo, com o nome do próprio subsistema, ou seja, o *Black Box*.

5.1 Implementação

O principal objetivo deste projeto é obter um sistema para a resolução 8K 60 *fps*, mas este foi desenhado para ser escalável para outras resoluções desde o início do desenvolvimento do projeto. A implementação deste módulo tem em conta que este deve funcionar para diferentes resoluções e *frame rates*.

5.1.1 Módulo *Black Box*

O retângulo preto depende da resolução e *frame rate*. Portanto, o número de linhas e de colunas, o número de *frames* e a indicação se o *frame rate* é inteiro ou fracionário são entradas deste módulo. Estes sinais estão representados na figura 5.1 pelas entradas *columns*, *lines*, *frames* e pelo sinal *integer*, que assumindo o valor 1 significa que o *frame rate* escolhido para a configuração é inteiro e quando assume zero é fracionário.

Sendo assim, é possível configurar através de *software* com os seguintes parâmetros:

- **Resolução:** 1080p, 4K, 8K;

- ***Frames* por segundo:** 23, 30, 50, 60;
- **Tipo de *frame rate*:** inteiro ou fracionário.

O sistema foi idealizado para suportar apenas estes *frame rates* e resoluções, já que para a empresa só tem interesse o desenvolvimento destas grandezas, mas a qualquer momento é possível adicionar mais resoluções e *frame rates* ao projeto.

Este módulo apresenta também uma entrada *AXI4-stream* escravo por onde recebe a *stream* e uma mestre por onde envia a *stream* com a adição do elemento gráfico para o próximo subsistema.

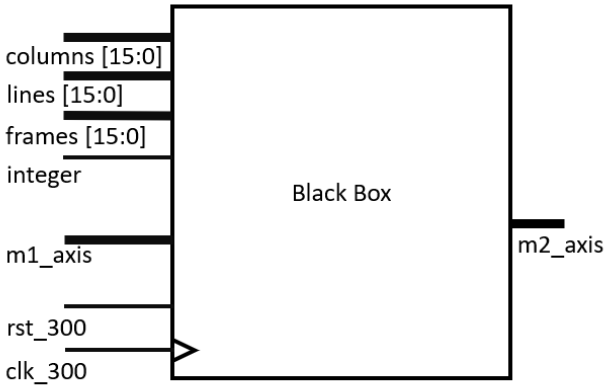


Figura 5.1: Diagrama de blocos da parte do sistema *Black Box*.

5.1.1.1 Cálculo da posição do retângulo

Como se pretende que este retângulo se encontre sensivelmente no meio da imagem e com dimensão de 1/16 da resolução horizontal da *frame* por 1/16 da resolução vertical da mesma, foi necessário calcular entre que pixeis da *frame* se pretende que ele apareça dependendo da resolução. Na tabela 5.1 é possível observar quais os valores que derivam das seguintes equações em que a equação 5.1 foi utilizada para calcular a coluna inicial do retângulo, a equação 5.2 para a coluna final, a equação 5.3 para a linha inicial e a equação 5.4 para a linha final:

$$Coluna\,inicial = resolução\,horizontal \times \frac{15}{32} \tag{5.1}$$

$$Coluna\,final = resolução\,horizontal \times \frac{17}{32} \tag{5.2}$$

$$Linha\,inicial = resolução\,vertical \times \frac{15}{32} \tag{5.3}$$

$$Linha\,final = resolução\,vertical \times \frac{17}{32} \tag{5.4}$$

Tabela 5.1: Posição do retângulo preto.

Resolução	Coluna inicial	Coluna final	Linha inicial	Linha final
1080p	900	1020	506	574
4K	1800	2040	1013	1148
8K	3600	4080	2025	2295

Os valores apresentados na tabela 5.1 são sempre atribuídos quando existe o *reset* do sistema em função dos parâmetros que recebe do módulo de configuração.

5.1.1.2 Adição do retângulo

O funcionamento interno do módulo *Black Box* pode ser descrito pelo multiplexador da figura 5.2, em que consoante uma condição vai colocar no barramento de dados da interface *Axi4-stream m2_axis* o valor presente no barramento de dados da interface *AXI4-stream m1_axis* ou os bit representativos da cor preta no formato YUV 422 8-bit.

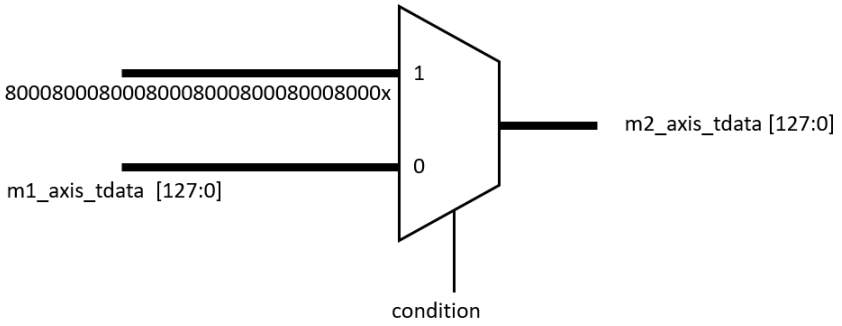


Figura 5.2: Multiplexador do módulo *Black Box*

Para tal é utilizado um contador de linhas, um contador de colunas e um contador de *frames*. Os primeiros dois permitem saber em que parte da *frame* o sistema se encontra quando está a receber dados (se estiver entre os valores calculados na tabela 5.1, está na posição em que deve estar o retângulo). Já o contador de *frames* permite contar as *frames* até a um limite que é definido pela entrada *frames* e sempre que atingir esse limite significa que passou um segundo, e o retângulo deve aparecer se não estivesse visível e desaparecer em caso contrário. Ou seja, para o retângulo preto aparecer na *stream* a entrada de seleção *condition* do multiplexador tem de ser verdadeira. Esta é verdadeira nos segundos ímpares, quando os contadores de colunas e linhas estão entre os valores apresentados na tabela 5.1. Existe ainda mais um detalhe importante relativo ao contador de colunas, pois como estão a ser utilizados 8 pixeis por *clock*, a cada ciclo de relógio chegam pixeis de 8 colunas diferentes ao módulo e portanto este contador incrementa de 8 em 8 posições.

Perante um *frame rate* fracionário, o sistema tem de se comportar de maneira um pouco diferente, ou seja, o contador de *frames* tem de saltar alguns valores, que são os apresentados na

secção 2.2.3. Recapitulando, de um em um minuto o ciclo de contagem do contador de *frames* no primeiro segundo vais ser: ... 28, 29, 2, 3 ... em vez de ... 28, 29, 0, 1, 2, 3 ... exceto nos minutos múltiplos de 10 para o caso dos 29.97 *fps*. Para o caso dos 59.94 *fps* e nas mesmas condições, o ciclo de contagem é ... 58, 59, 4, 5 ... em vez de ... 58, 59, 0, 1, 2, 3, 4, 5 ... .

5.2 Teste

Nesta fase do projeto já existia o subsistema *2SI* e portanto já era possível observar as *frames* divididas por este subsistema, no entanto como estes capítulos seguem o fluxo de dados, esses testes apenas vão ser apresentados no capítulo 7 onde é descrito o módulo *2SI*. Portanto, neste capítulo apenas são apresentados os testes que foram realizados à saída do módulo *black box*.

Estes testes foram novamente dois:

- Simulações comportamentais do circuito e guardar os dados das *frames* num ficheiro binário;
- Colocar um VDMA à saída do módulo *Black Box* e implementar na FPGA para observar os dados que vão para o VDMA.

Para ambos os testes foram realizadas iterações com todos os *frame rates* e resoluções para os quais se pretende que o sistema opere. Como foi realizado um plano de testes para o sistema que permite o reaproveitamento e utilização dos mesmos testes sem grandes alterações independentemente do subsistema do projeto, para testar este subsistema apenas foi necessário fazer as seguintes alterações:

- Quando foram realizadas simulações comportamentais, no *design* os sinais da interface mestre do módulo foram colocados como *outputs*.
- No teste como o VDMA, o que mudou, foi a posição do VDMA, que em vez de estar a guardar dados que passam a interface m1_axis, estava a guardar os que passam pela m2_axis.

Também foram realizados testes através da análise das formas de ondas nos *frame rates* fracionário, para comprovar o valor dos contadores, mas os resultados destes testes apenas são apresentados no capítulo 7. Isto deve-se ao facto de os testes serem iguais aos desenvolvidos no capítulo 7, e faz mais sentido serem abordados nesse capítulo relacionado com o *Timecode*, pois torna-os mais fáceis de compreender.

5.3 Resultados

Na figura 5.3 é possível observar a vigésima sexta *frame* da *stream* de vídeo guardada no VDMA para uma configuração de 8K 25 *fps*. A vigésima sexta *frame* de uma *stream* a 25 *fps* é a primeira *frame* do segundo segundo. Este (segundo nº 1), é ímpar e como foi mencionado anteriormente, o retângulo preto deve aparecer em todas as *frames* desta paridade de segundos.

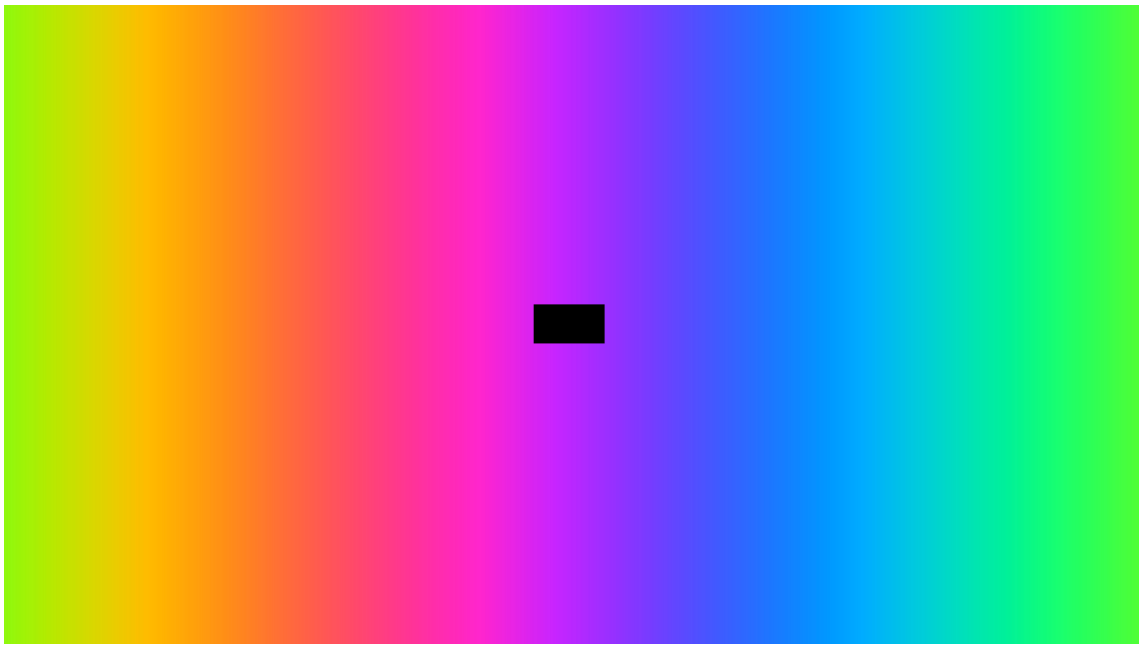


Figura 5.3: *Frame* 8K retirada do VDMA com o padrão "Color Sweep"para o teste do *Black Box*.

Neste capítulo foi apresentada a implementação, teste e resultados do subsistema *Black Box*, mas no capítulo 8, em que se fala do sistema em geral são demonstrados mais alguns testes que envolvem este subsistema a interagir com os outros.

Capítulo 6

Subsistema de geração do *Timecode*

O outro elemento gráfico adicionado ao padrão de vídeo é o *timecode*. Este elemento gráfico é construído no subsistema também chamado de *Timecode*. Este subsistema pode ser decomposto em duas partes principais que realizam duas funções:

- Adição do fundo onde se vão sobrepor os números;
- Adição dos números à *frame*.

Na figura 6.1 é possível observar o *timecode* de uma *frame* e distinguir estas duas parte.

Figura 6.1: Exemplo do *timecode* de uma *frame*.

6.1 Implementação

Este subsistema, representado no diagrama de blocos da figura 6.2 é composto pelo módulo *White box*, pelo módulo *Numbers* e por quatro instâncias do módulo *16-bit Divider*. O módulo *white Box* apresenta o sinal *m2_axis* ligado à sua interface *AXI4-stream* escravo e o m3_axis à sua interface mestre. Para além destas interfaces, este módulo também recebe a resolução pelas entradas *lines* e *columns*. Este módulo não necessita de conhecer o *frame rate*, uma vez que está estático na figura, ou seja, quando adicionado ao *design*, os dados provenientes do *White Box* são sempre adicionados à *stream* da mesma maneira (está sempre presente e na mesma posição).

Quanto ao módulo *Numbers*, este apresenta o sinal *m3_axis* ligado à sua interface escravo *AXI4-stream* e o m4_axis à sua interface mestre. Este módulo também recebe os sinais *columns*,

lines, *frames* e o *integer*. Para realizar a divisão binária, este módulo está conectado com 4 instâncias do *16-bit Divider*. O objetivo do *16-bit Divider* é realizar a divisão binária do sinal *dividend_n [15:0]* por 10 quando o sinal *start_n* vai a 1, e apresentar o quociente no *result_n[15:0]* e o resto em *rest_n[15:0]* (em que o *n* pode assumir o valor de 1 a 4 dependendo da instância do *16-bit Divider* ao qual o sinal está ligado). Logo que acaba o cálculo e tem os resultados prontos, o sinal *finish_n* é colocado a 1 para indicar ao módulo *Number* que já terminou. Cada instância do *16-bit Divider* é responsável por cada um dos 4 pares de dois dígitos da figura 6.1. O algarismo da direita de um par de dígitos corresponde ao resto e o da esquerda ao quociente.

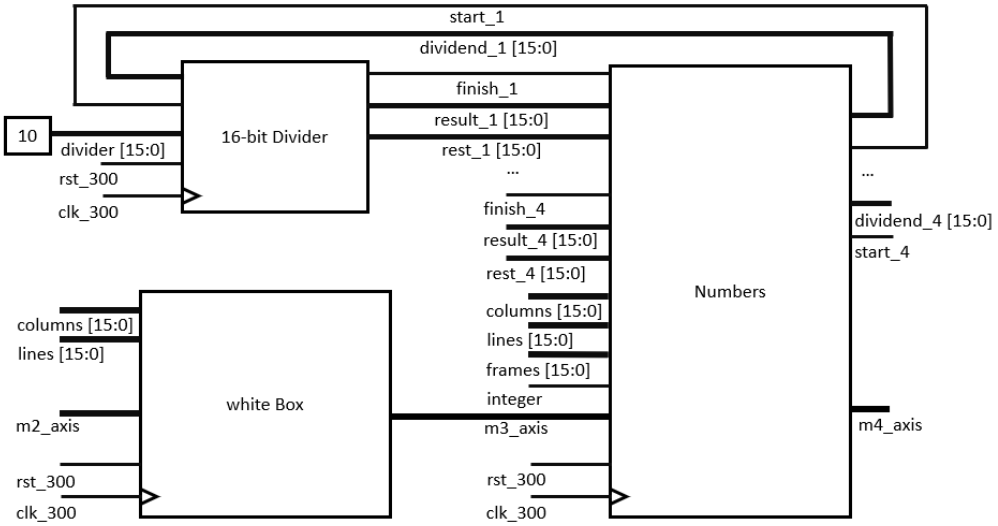


Figura 6.2: Diagrama de blocos do subsistema *Timecode*.

Devido a este subsistema apresentar duas partes distintas e separáveis, durante a implementação, a sua funcionalidade vai ser demonstrada de forma separada.

6.1.1 Fundo

6.1.1.1 Cálculo da posição do fundo

A posição pretendida para o fundo é um pouco abaixo do retângulo preto, mas também centrada horizontalmente. Para calcular o valor das suas posições apresentadas na tabela 6.1 foram utilizadas as seguintes fórmulas:

$$Coluna\,inicial = \frac{resolução\,horizontal}{1920} \times 688 - 1 \tag{6.1}$$

$$Coluna\,final = Coluna\,inicial + \frac{resolução\,horizontal}{1920} \times 544 \tag{6.2}$$

$$Linha\,inicial = \frac{resolução\,vertical}{1080} \times 830 - 1 \tag{6.3}$$

$$Linha\,final = Linha\,inicial + \frac{resolução\,vertical}{1080} \times 132 \tag{6.4}$$

$$Espessura\,da\,borda = \frac{resolução\,vertical}{1080} \times 4 \tag{6.5}$$

Os valores apresentados na tabela 6.1 são sempre atribuídos quando existe o *reset* do sistema, em função dos parâmetros de entrada vindos do módulo de configuração. No entanto, os valores apresentados na tabela para o início e fim das colunas incluem a borda, para calcular os valores iniciais do retângulo branco é somar o valor da borda para as linhas e colunas, e para as finais subtrair este mesmo valor.

Tabela 6.1: Posição do fundo.

Resolução	Coluna inicial	Coluna final	Linha inicial	Linha final	Espessura da borda
1080p	687	1231	829	961	4
4K	1375	2463	1559	1923	8
8K	2751	4928	3319	3847	16

6.1.1.2 Adição do fundo

A adição do fundo é semelhante à adição do retângulo preto abordado no capítulo anterior, mas no entanto este objeto apresenta duas diferenças derivadas da sua maior complexidade:

- Composto por duas cores (branco para o fundo e preto para a borda);
- A borda é bastante fina, e é necessário mais pormenor.

Para resolver este problema, em vez de um multiplexador, são utilizados 4. Ou seja, em vez de um multiplexador tratar de 8 pixeis de cada vez, só trata de 2. Isto permite, que em cada ciclo de relógio os 8 pixeis que chegam ao módulo possam assumir cores diferentes (2 a 2) em vez de terem todos a mesma cor. Na figura 6.3 é possível verificar um exemplo, para o caso de uma *frame* 1080p, em que os 8 pixeis que saem do módulo num dado ciclo de relógio assumem cores diferentes 2 a 2. Os da direita são do fundo, os do meio da borda e os da esquerda do padrão que o módulo recebeu.

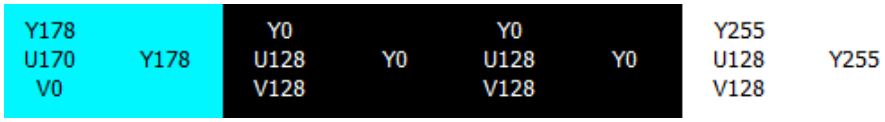


Figura 6.3: Exemplo de um grupo de 8 pixeis que saiem do módulo *white Box* para a resolução 1080p.

Na figura 6.4 é possível observar os 4 multiplexadores, que apresentam 3 entradas, os dados do barramento de entrada *AXI4-stream* do módulo, a cor branca e a cor preta. Este módulo também apresenta um contador de linhas e um de colunas que conta de 8 em 8. As 4 condições que selecionam as entradas de seleção dos multiplexadores vão comparar o valor dos contadores com os limites (tabela 6.1) entre os quais deve aparecer o fundo, borda ou os dados do barramento de entrada *AXI4-stream*. Cada uma delas apresenta uma constante diferente adicionada ao contador de colunas para tratar dos 4 grupos diferentes de 2 pixeis. A constante adicionada na "*condition 1*"é -7 o que permite tratar do primeiro e do segundo pixel, na "*condition 2*"é -5 para tratar do terceiro e quarto pixel, na "*condition 3*"é -3 para o quinto e sexto pixel e na "*condition 4*"é -1 para o sétimo e oitavo pixel.

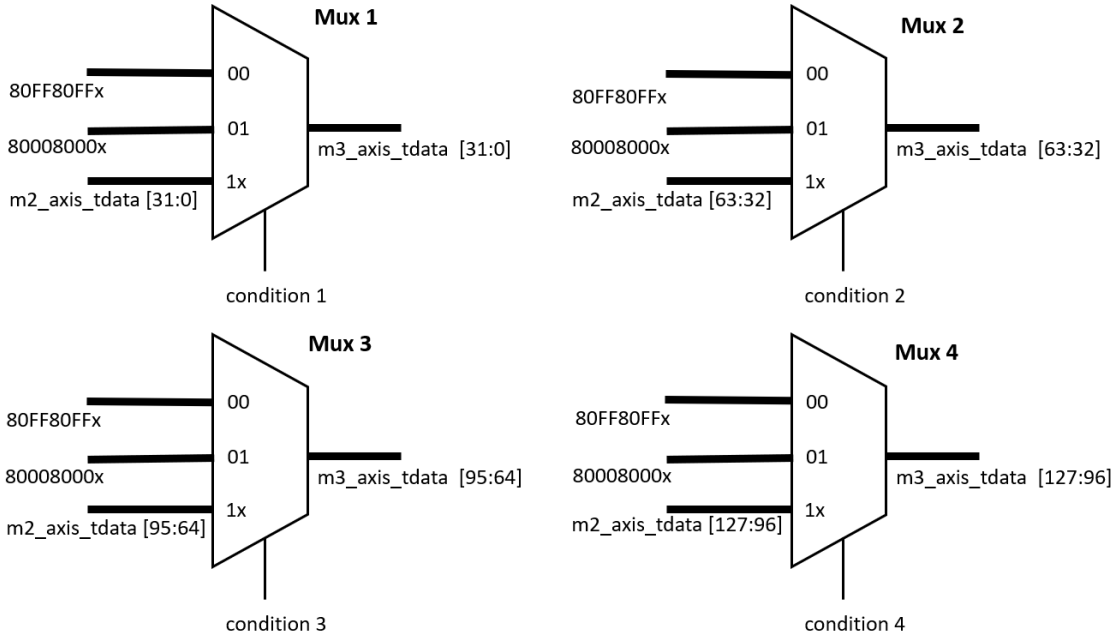


Figura 6.4: Multiplexadores do módulo *White Box*.

Juntando a informação das figuras 6.4 e 6.3, pode observar-se que o *Mux 1* é responsável pelos dois pixeis azuis, os *Mux 2* e *Mux 3* pelos pretos (borda) e o 4 pelos brancos (fundo). Os outros sinais da interface *AXI4-stram* são diretamente atribuídos à saída do módulo.

6.1.2 Números

A parte do subsistema responsável por adicionar os números sobre o fundo é composta pelo módulo *Numbers* e pelos 4 divisores binários de 16 bit.

6.1.2.1 Posição dos números

O cálculo da posição dos números utiliza uma metodologia semelhante ao cálculo da posição do retângulo preto e do fundo em que as constantes dependiam do valor da resolução, no entanto os

números são elementos gráficos muito mais complexos, com muitas mais constantes. Nesta parte do sistema não vão ser detalhados os valores para as constantes de posicionamento dos elementos gráficos, por ser demasiado extenso e o foco principal é o método utilizado para representar os números.

Para adicionar os números, o método utilizado passa por utilizar um mecanismo semelhante a um *display* de 7 segmentos. São utilizados 8 *displays* de 7 segmentos, um para cada algarismo do *timecode*. Na figura 6.5 é possível observar o *display* de 7 segmentos que foi desenhado. Cada quadrícula representa uma matriz de 2 x 2 pixeis para a resolução 1080p. Para um melhor aspeto visual, cada segmento é composto por 3 retângulos sobrepostos, e existem segmentos horizontais e verticais. Ou seja, para cada *display* de 7 segmentos, na realidade existem 3 x 7 = 21 segmentos.

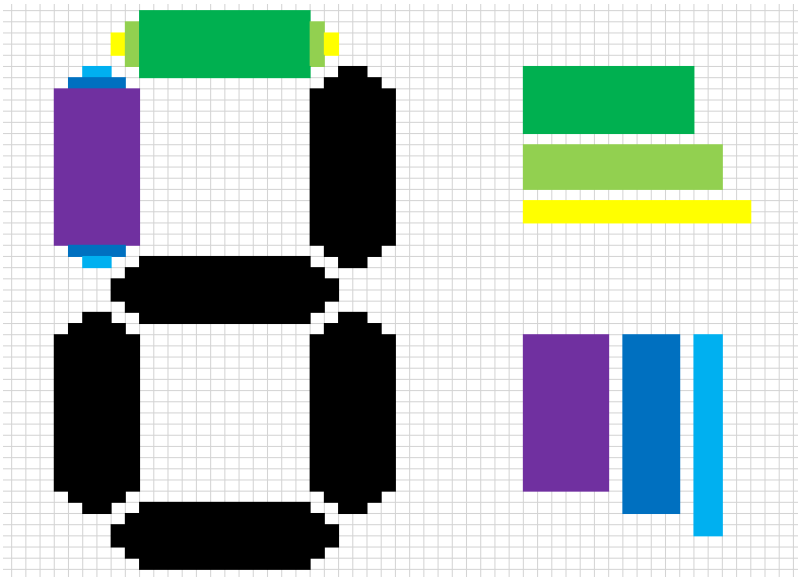


Figura 6.5: Esquema de um *display* de 7 segmentos.

Os 8 *displays* são construídos a partir de um primeiro, ou seja, são iguais ao primeiro mais a soma de uma constante na posição horizontal para cada um deles, resultando o esquema da figura 6.6. Existem também 6 pontos para separar os números que também são construídos por 3 retângulos para um melhor aspeto visual.

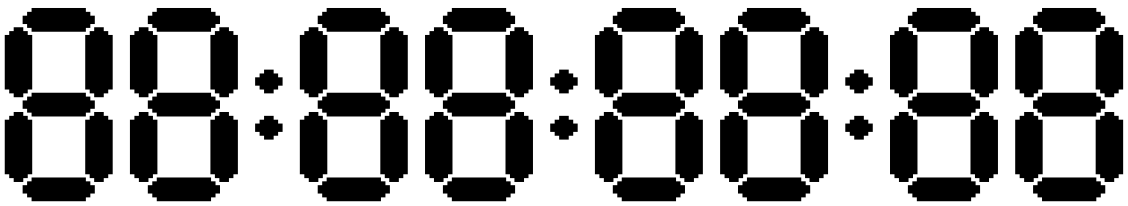


Figura 6.6: Esquema da matriz de *displays*.

Se olharmos para cada retângulo como um elemento gráfico, este módulo é responsável por adicionar 3 x 7 x 8 (dos números) + 3 x 6 (dos pontos) = 186 elementos gráficos.

6.1.2.2 Adição dos números

Como acontece em alguns módulos anteriores, este também apresenta um contador de linhas e um contador de colunas que conta de 8 em 8. No entanto, este módulo apresenta também um contador de *frames*, um de segundos, um de minutos e um de horas. Quando chega uma nova *frame* estes contadores são atualizados e posteriormente os sinais *start_n* (*n* varia)são colocados a 1 durante um ciclo de relógio, para indicar às instâncias do módulo *16-bit Divider* que podem iniciar a divisão por 10 dos valores que recebem na variável *dividen_n*. Cada um destes módulos é responsável por fazer a divisão de cada um dos 4 contadores mencionados em cima para obter os valores dos algarismos das unidades e das dezenas para cada um dos contadores .

Esta operação demora alguns ciclos de relógio desde que chega o sinal que indica o início de uma nova *frame*. Este tempo acaba por não ser significativo devido ao módulo *Numbers* só precisar dos resultados das divisões a partir do meio da *frame*, pois a *stream* chega linha a linha e estes elementos gráficos só se encontram para lá do meio da *frame*. Os resultados da divisão de cada contador vão fornecer os algarismos de cada conjunto de dois *displays*. Por exemplo, no conjunto de dois *displays* mais à direita, o algarismo mais à direita é o resto da divisão por 10 do contador de *frames* e o mais à esquerda o seu quociente. Em função dos resultados obtidos, são utilizadas variáveis auxiliares para indicar quais os segmentos que têm de estar ativos para um determinado algarismo.

Os números são elementos gráficos que apresentam uma única cor e que são tratados de dois em dois pixeis tais como no caso do bloco *White Box*. Sendo assim, esta parte do subsistema comporta-se como é ilustrado na figura 6.7 em que existem 4 multiplexadores.

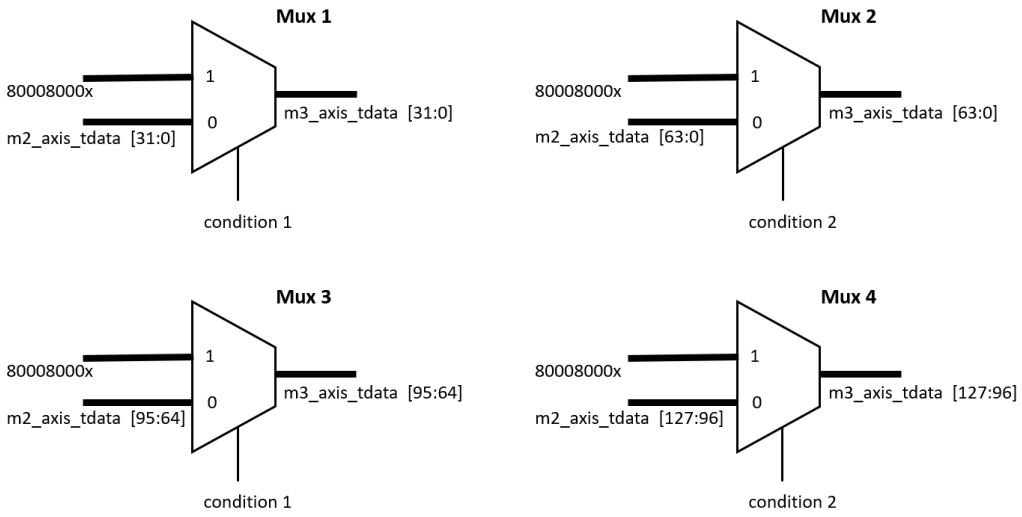


Figura 6.7: Multiplexadores do módulo *Numbers*.

Cada multiplexador trata de dois pixeis e são somadas as mesmas constantes aos contadores de colunas que no caso dos multiplexadores do *white Box* (-7 para o primeiro e segundo, -5 para o terceiro e quarto, -3 para o quinto e sexto, -1 para o sétimo e oitavo). Como os números só apresentam uma cor, só são necessárias duas entradas por para o multiplexador. Sempre que as condições que o comandam as entradas de seleção forem verdadeiras, eles colocam nas suas saídas a cor preta, senão colocam nas suas saídas os dados que recebem pela interface *AXI4-stream* escravo do módulo.

Para sumarizar, as condições que comandam os multiplexadores verificam-se quando os dois pixeis pelo qual um multiplexador é responsável pertencem aos segmentos, e se numa determinada *frame* um desses segmentos está ativo. O barramento de dados é trabalhado pelos multiplexadores e os restantes sinais da interface *AXI4-stream* de entrada (escravo) são diretamente atribuídos à interface que se encontra à saída do módulo (mestre).

6.2 Teste

Para testar o subsistema foi utilizada a mesma metodologia de teste usada para os subsistemas dos capítulos 4 e 5. Ou seja, primeiro com simulações comportamentais à saída do módulo que se pretende analisar, e depois com um VDMA após a implementação na FPGA . Tendo em conta que este se encontra depois do *Black Box* e antes do *2SI*, ao testar o subsistema deverá aparecer na *frame* o retângulo preto e também o *timecode*.

Como este subsistema foi implementado em duas etapas, também existiram duas etapas de teste. Os testes à saída do *White Box* e os testes à saída do *Numbers*. Na primeira etapa de teste foram analisados os dados que saíam do módulo *White Box* ainda sem o *timeode*, e na segunda etapa foram analisados os dados à saída do módulo *Numbers* já com todos os elementos gráficos adicionados no padrão (retângulo preto, fundo do *timecode* e o próprio *timecode*).

Como já foi mencionado no capítulo 5, foram realizados testes a nível comportamental do circuito para verificar se no caso dos *frame rates* fracionários o sistema está a funcionar como esperado. Este teste consistiu em configurar a resolução para um valor pequeno (64 x 64) apenas para exercitar os contadores de *frames*, segundos, minutos e horas, e verificar se estão a funcionar como se esperava, pois o seu comportamento é diferente de quando são *frame rates* inteiros. Foi utilizada esta resolução para tornar a simulação mais rápida, pois apenas interessa verificar se os contadores estão a funcionar corretamente.

6.3 Resultados

Na figura 6.8 é possível observar os resultados do teste para a primeira etapa mencionada na secção anterior, em que a *frame* apresenta o fundo para o *timecode*. A imagem da figura não apresenta o retângulo preto ativo pois representa *frame* do segundo 0, em que o retângulo preto não está ativo. Esta *frame* foi retirada de um VDMA que guarda as *frames* que passam pela interface *AXI4-stream ms3_axis* representada no diagrama de blocos da figura 6.2.

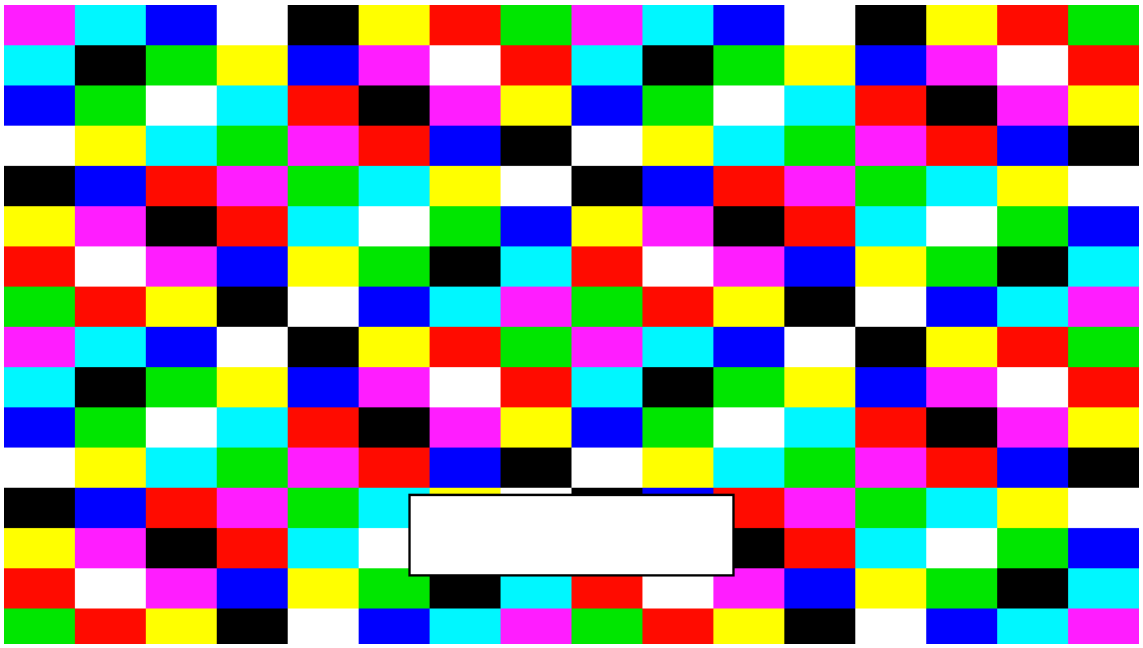


Figura 6.8: *Frame* 8K retirada do VDMA com o padrão "Taratan Bars"para o teste do fundo para os números.

Por sua vez na figura 6.9 é possível observar o teste feito com o VDMA a guardar as *frames* que passam pela interface *m4_axis*. Neste teste é utilizada a vigésima sexta *frame* de uma configuração 8K 25 *fps*.

Figura 6.9: *Frame* 8K retirada do VDMA com o padrão "Tartan Bars"para o teste dos números.

A imagem apresentada na figura anterior representa uma *frame* já com todos os elementos presentes antes de ser enviada para o subsistema responsável por dividi-la.

Quanto ao teste relacionado com os contadores, no caso do *frame rate* fracionário verifica-se que quando estamos perante a passagem para um novo minuto não múltiplo de 10, para o caso dos 29.97 *fps*, existe um salto de duas unidades, como era de esperar (figura 6.10). Já nos minutos múltiplos de 10, também como era de esperar, na figura 6.11 podemos verificar que o salto não existe e o contador de *frames* ao iniciar o minuto 10 passa de 30 para 0, exatamente como deveria ocorrer para o caso de um *frame rate* inteiro.

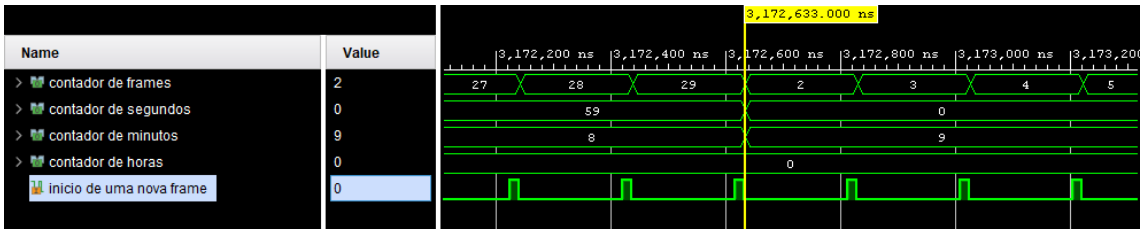


Figura 6.10: Transição do minuto 8 para o 9 como *frame reate* a 29.97.

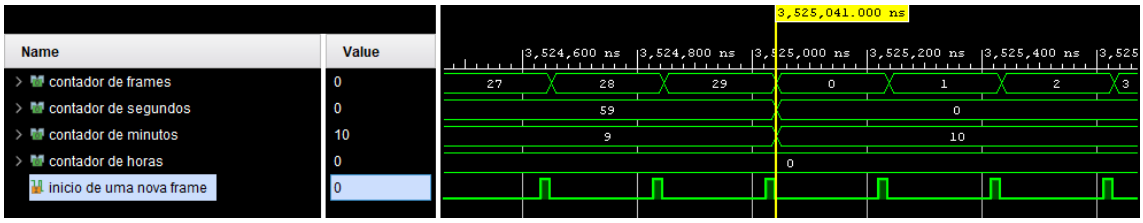


Figura 6.11: Transição do minuto 9 para o 10 como *frame reate* a 29.97.

Este subsistema é o último da cadeia de dados antes de chegar ao subsistema responsável por fazer a divisão da *stream* pelos diversos *links*.

Capítulo 7

Subsistema que implementa o método 2SI

O último subsistema que é abordado antes da integração, é o responsável pela divisão da *stream* pelos diversos *links* de dados. Este subsistema é composto por duas partes, uma que organiza os dados para serem enviados e outra realiza a divisão pelos diversos *links*.

7.1 Implementação

Este subsistema está representado no diagrama de blocos da figura 7.1 e é constituído pelo módulo *Data Organizer*, 4 instâncias do IP *Axi Data FIFO* e pelo *Data Distributor*.

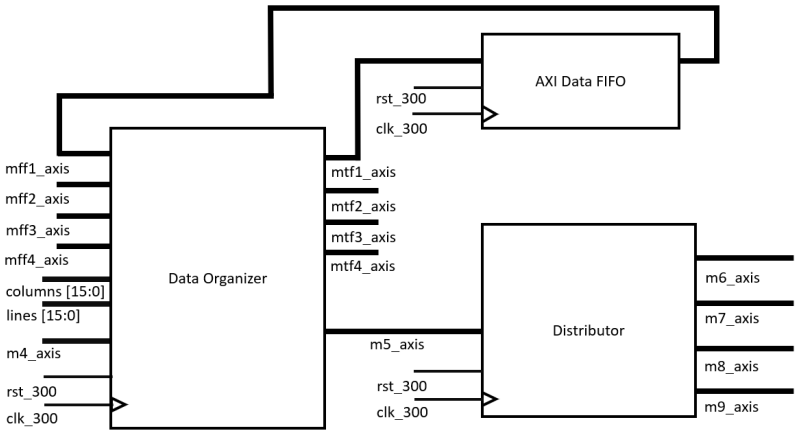


Figura 7.1: Diagrama de blocos do subsistema *2SI*.

A *stream* chega ao *Data Organizer* pela interface *m4_axis* e este envia para as 4 *FIFO* (1 linha da *stream* para cada FIFO ciclicamente) também por *AXI4-stream*. Cada FIFO está ligada a um conjunto de sinais *mtf(n)_axis* e *mff(n)_axis* do módulo *Data Organizer* (ambos *AXI4-stream*). Quando o *Data Organizer* precisa de dados para enviar para o *Distributor* pela interface *m5_axis*,

este vai buscar os dados às tais FIFOs. Por sua vez, o *Distributor* distribui os dados que chegam, por 4 interfaces *AXI4-stream* (uma para cada *link* SDI).

7.1.1 Parte que implementa o método 2SI

7.1.1.1 Latência e Memória caraterísticos do método

Como já foi referido no capítulo 2, o método 2SI apresenta menos latência de processamento que o *square division*. A latência de processamento do método implementado é, aproximadamente, o tempo de transmissão de duas linhas da imagem, pois para realizar este método são necessárias duas linhas de cada vez. Os dados chegam ao módulo linha à linha, e portanto o módulo *Data Organizer* só inicia a transmissão dos dados para o *Distributor* duas linhas depois de receber o primeiro SOF. Para tal é necessário guardar a informação em FIFOs. São utilizadas 4, pois quando a primeira e a segunda linha estão a ser enviadas para o *Distributor*, o *Data Organizer* já está a receber a terceira e a quarta e assim sucessivamente. Isto é a única forma de conseguir receber e enviar todos os dados, pois o *Data Organizer* recebe uma linha de cada vez, mas envia duas de cada vez. A cadência de dados no envio e receção é a mesma, mas no entanto, no envio estão a ser enviados 4 pixeis de cada linha, enquanto na receção, são recebidos 8 de apenas uma linha, o que implica o dobro do tempo para enviar uma linha completa comparativamente com o tempo da receção. Quando o sistema recebe uma nova linha ímpar, envia a primeira metade das duas linhas anteriores e quando recebe uma par, envia as últimas metades.

Isto implica que os sinais da interface *m5_axis* tenham de ser atrasados cerca de duas linhas relativamente aos da *m4_axis*. Para tal foi necessário implementar duas máquinas de estados, uma para o SOF e outra para os restantes sinais.

7.1.1.2 Máquina de estados para o SOF

A máquina de estados presente na figura 7.2, encarrega-se de gerar o SOF da interface *m5_axis* do bloco *Data Organizer* em função da *m4_axis*. O sinal SOF da interface *m5_axis* para estar duas linhas atrasado, têm de primeiro acontecer alguns eventos na interface *m4_axis*. Após receber o sinal *s_SOF*(SOF da interface *m4_axis*), o sistema passa do estado *Hidle* para o *SOF_slave*. Posteriormente, quando chega o sinal *s_tvalid*, o sistema vai para o estado *Line 1* em que recebe a primeira linha que acaba quando o sistema recebe o sinal *s_tlat_delay* (*tlast* atrasado um ciclo de relógio) e vai para o estado *End of Line 1*. Quando este sinal vai a 0 significa que estamos na segunda linha (estado *Line 2*) e quando vai novamente a 1, a terceira linha está para chegar e então o sistema vai para o estado *SOF_master* em que ativa o sinal SOF da interface *m5_axis* até o sinal *s_tlast_delay* voltar novamente a 0. Para esta máquina de estados é utilizado preferencialmente o sinal *tlast* da interface *m4_axis* para desencadear os eventos em vez do *tvalid*, pois durante a transmissão de uma linha, este pode alternar (estar a o ou a 1) consoante faça ou não falta dados no módulo seguinte da cadeia de dados (só envia quando *ready* está a 1). Já o *tlast* desta interface, tem um comportamento mais previsível (vai a 1 no último ciclo de dedos de uma *frame* e volta a 0 no ciclo antes de serem novamente enviado dados). É utilizado o *tlast* atrasado um ciclo de

relógio, pois como foi referido do capítulo 2, no protocolo *AXI-stream* para vídeo, no primeiro ciclo de dados, o sinal SOF ainda tem de estar ativo.

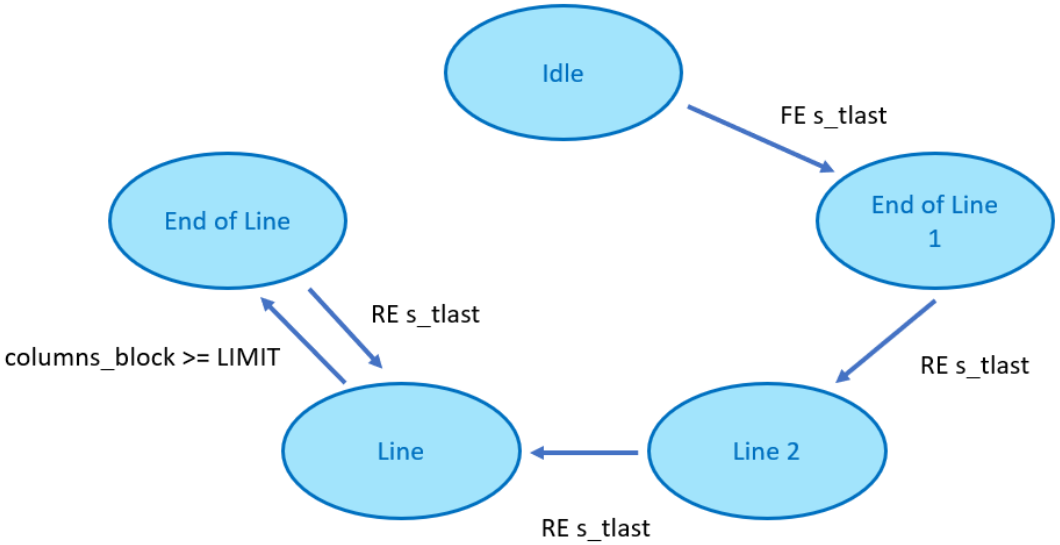


Figura 7.2: Máquina de estado para gerar o SOF.

Na figura 7.3 é possível observar, que na terceira linha de uma *frame* quando os dados começam a ser enviados e o *tvalid* da interface mestre está ativo, o SOF do mestre ainda está um ciclo de relógio a 1. Por este motivo é que tem de se usar o sinal *tlast* do escravo atrasado um ciclo de relógio para desencadear o fim do *SOF* da interface *m5_axis*. Isto permite cumprir o requisito apresentado no capítulo 2 relativo à utilização de *AXI4-stream* para transmissão de vídeo.

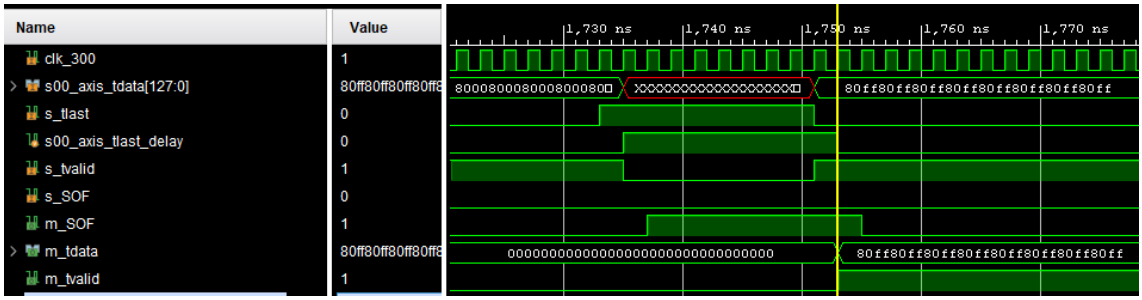


Figura 7.3: Sinais para gerar o SOF.

Na próxima máquina de estados, que é a máquina que comanda os sinais *tdata*, *tlast* e *tvalid*, é possível perceber como estes sinais são processados para estarem de acordo com o SOF.

7.1.1.3 Máquina de estados para os restantes sinais

A máquina de estados apresentada na figura 7.4 é responsável por tratar dos sinais *tdata*, *tlast* e *tvalid* da interface *m5_axis*. Mas antes de entrar na explicação da máquina de estados vão ser

abortados alguns aspetos importantes.

Neste módulo também existe um contador de colunas para um conjunto de um par de linhas que conta de oito em 8 e um contador de linhas que conta as linhas ciclicamente de 0 a 3. O limite até quando o contador de colunas tem de contar é achado através do parâmetro de entrada resolução horizontal. A contagem de linhas tem o objetivo de permitir ao sistema saber para que FIFO têm de enviar os dados, quando chega uma nova linha. Quando chega uma linha esta é enviada pela interface mtf(n)_axis. Este contador também permite saber a que FIFO o módulo terá de ir buscar dados. Como existe um *delay* de duas linhas, quando o contador estiver em 0 ou 1, e no estado *Line* da máquina de estados da figura 7.4, o módulo vai buscar dados às FIFOs 2 e 3, e quando está em 2 ou 3 vai buscar às 0 e 1.

A máquina de estados da figura 7.4 passa do estado *Hidle* para o *End of Line 1* no FE do sinal *s_last* da interface *m4_axis*, que significa que a primeira linha da *stream* já chegou ao fim. Quando houver um RE de *s_last* a máquina de estados vai para o estado *Line 2* e no próximo RE desta variável vai para o estado *Line* que representa o momento a partir do qual é necessário começar a enviar linhas para o módulo *Distributor*. Quando o contador de colunas para um conjunto de duas linhas (columns_block) atinge o limite, esta máquina de estados vai para o estado *End of line* e volta ao estado *Line* no RE do *s_last* da interface *m4_axis*.

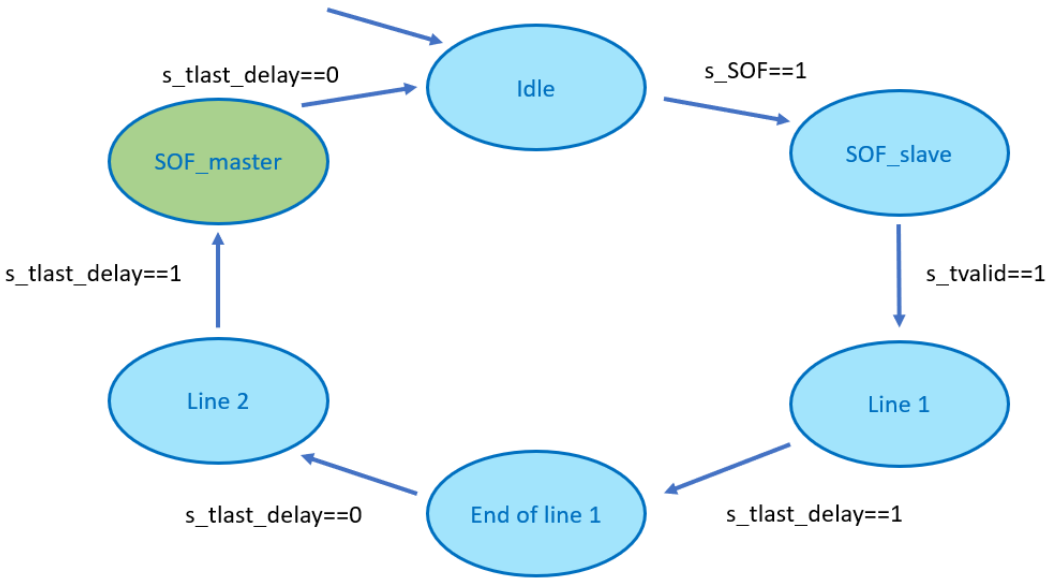


Figura 7.4: Máquina de estado para gerar o SOF.

O sinal *t_valid* é ativo quando o módulo está no estado *"Lines"* e neste momento são também atribuídos os dados guardados nas FIFO ao *t_data*. Como o sinal *tlast* tem de estar ativo no último ciclo de relógio de uma linha, este é colocado a 1 neste mesmo instante do estado *Lines* e durante o estado *End of Line*.

Pelas formas de onda da figura 7.5 é possível observar o comportamento do subsistema, que só a partir da segunda linha da primeira *frame* é que começa a enviar dados que vai buscar às FIFOs para o *Distributor*. Existe mais um detalhe que ainda não foi mencionado. O barramento de dados da FIFOs é de 128 bit, ou seja, quando se envia dados para uma FIFO, como se utiliza um destes módulos de cada vez, não são necessários cuidados adicionais, dado que o barramento de dados da interface *AIX4-stream* escravo *m4_axi* também é de 128. No entanto, quando se vai buscar dados às FIFO, esta operação é realizada a duas FIFOs de cada vez, ou seja, vão se buscar 256 bit e só se pode enviar pela interface *AIX4-stream m5_axi* 128. Para resolver este problema, quando são necessários dados para enviar por *m5_axi*, uma vez o sistema vai buscar 256 bit às FIFOs, envia 128 e guarda 128 numa variável auxiliar e outra vez não vai buscar às FIFOs, e usa-os da variável auxiliar, e assim sucessivamente (essencialmente, existe um pequeno *buffer*).

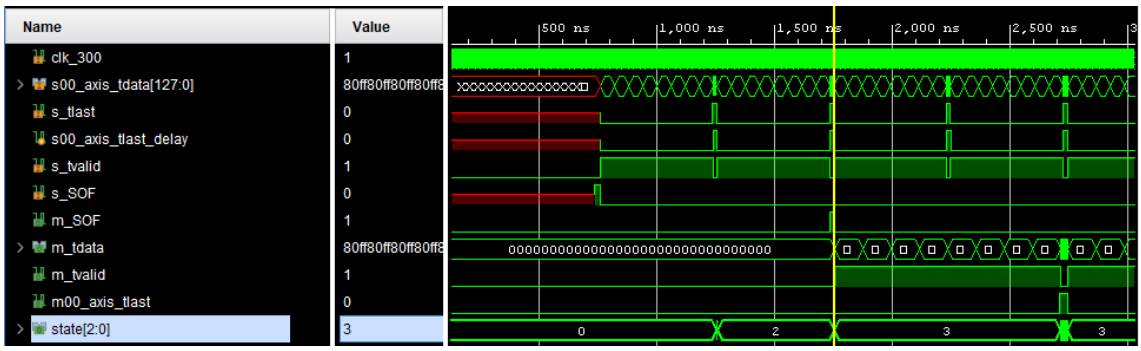


Figura 7.5: Sinais para gerar *tlast*, *tvalid* e *tdata* da interface mestre.

7.1.2 Distribuição dos dados pelos 4 *links*

Esta parte apenas é constituída pelo módulo *Distributor* que realiza a função de dividir os dados pelas 4 interfaces, para tal atribui o sinal *tlast*, *tvalid* e SOF da interface escravo às 4 mestres. Os sinais dos barramentos de dados das interfaces mestre têm de ter 64 bit, pois é exigido pelo módulo do subsistema que trata do envio para o SDI. Pata tal os bit 127 a 96 vão para o *link* 4, os 95 a 65 para o 3, os 63 a 32 para o 2 e os 31 a 0 para o 1 da seguinte forma:

Código Verilog 7.1: Código para os barramentod de dados do módulo *Distributor*

```
assign m6_axis = {24'b0, m5_axis[31:24],   2'b0, m5_axis[23:16],   2'b0, m5_axis
[15:8],    2'b0, m5_axis[7:0],    2'b0};

assign m7_axis = {24'b0, m5_axis[63:56],   2'b0, m5_axis[55:48],   2'b0, m5_axis
[47:40],   2'b0, m5_axis[39:32],  2'b0};

assign m8_axis = {24'b0, m5_axis[95:88],   2'b0, m5_axis[87:80],   2'b0, m5_axis
[79:72],   2'b0, m5_axis[71:64],  2'b0};
```

```
assign m9_axis = {24'b0 ,m5_axis[127:120], 2'b0, m5_axis[119:112], 2'b0, m5_axis
[111:104],  2'b0, m5_axis[103:96], 2'b0};
```

Os dois bit a 0 após cada componente de oito bit de um pixel serve para passar novamente para YUV 422 - 10 bit. Os 24 zeros no início de cada barramento apenas servem para ajustar os tamanhos dos próprios barramentos. O sinal *tready* da interface *m5_axis* só é colocado a 1 quando os *tready* de todas as interfaces mestres estão a 1.

7.2 Teste

Apesar de este ter sido o segundo subsistema implementado, nos testes e resultados, vão ser demonstrados testes do sistema com os subsistemas que adicionam o *timecode* e o retângulo preto.

Para o teste deste subsistema, seguiu-se a mesma metodologia utilizada nos subsistemas anteriores, através do *testbench* e dos VDMAs. No entanto, apesar de este subsistema estar dividido em duas partes, não faz sentido utilizar o *testbench* nem os VDMAs para testar a interface *AXI4-stream* entre o módulo *Data Organizer* e o *Distributor*, pois apesar de estes utilizarem *AXI-stream* para comunicar entre si, os dados que eles transmitem não conseguem ser diretamente associados a uma *frame*. Apesar de serem transmitidos 8 pixeis, cada par pertence a uma *frame* com 1/4 da resolução da original e a duas linhas diferente. Sendo assim só faz sentido analisar as interfaces *AXI* à saída do *Distributor*. Para tal foram realizadas as seguintes alterações em cada um dos teste:

- No *testbench* houve uma alteração para em vez de se analisar uma interface *AXI-stream* e guardar uma *frame* para um ficheiro binário, conseguir analisar 4 interfaces e guardar em 4 ficheiros diferentes, em que cada um adquire *frames* com 1/4 da resolução original. Desta maneira é possível testar a informação que vai para todos os *links* de uma vez;
- No *block design*, antes de implementar na FPGA, foram adicionados 4 VDMAs (1 para cada interface *AXI-stream* mestre do *Distributor*).

Para além destes testes, foi adicionado no *testbench* a funcionalidade de reconstruir a *frame* original emulando assim o processo que ocorre do lado do RX. Este processo passa por imprimir diretamente para um ficheiro os *links* 1 e 2 (linhas ímpares) e quando é recebido o sinal *tlast* imprimir os *links* 3 e 4 (linhas pares) que entretanto foram guardados em variáveis auxiliares. Desta simples forma é possível reconstruir a *frame* e validar que a divisão desta está a ser bem feita e não provoca erros na *stream*.

7.3 Resultados

Nesta secção vão ser apresentados os resultados para o caso do 1080p de maneira a perceber melhor o efeito do método 2SI. Na figura 7.6, é possível observar uma *frame* 960 x 540 obtida através da análise do *link* 1 no *testbench*.

Figura 7.6: *Frame* na resolução 960 x 540 obtida a partir do *testbench* (1 link através da resolução original 1080p).

Por sua vez, na figura 7.7, ao ampliar o *timecode* é possível observar, que apesar de existirem 4 *frames* com 1/4 da resolução original ao usar o 2SI, se olharmos só para uma *frame*, esta vai apresentar pior qualidade, o que era de esperar.

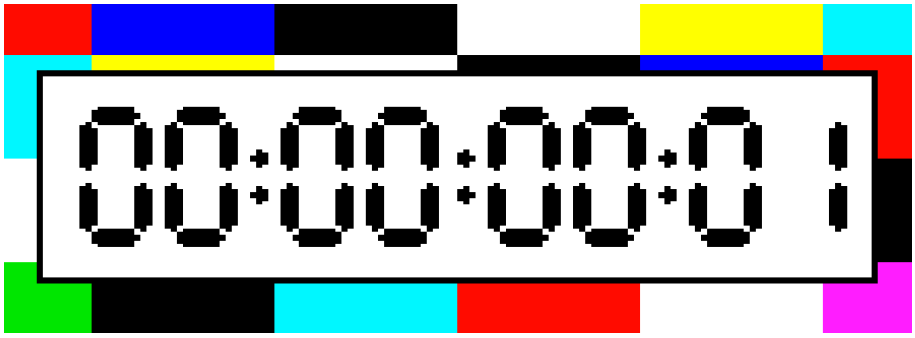


Figura 7.7: Ampliação do *timecode* numa *frame* 960 x 540.

Na figura 7.8, é possível observar uma *frame* reconstruida no *testbench*, que permite verificar que a imagem é igual à original com resolução 1080p, o que era desejado.

Figura 7.8: *Frame* na resolução 1080p obtida a partir do *testbench*.

Quanto aos testes realizados com os VDMAs, estes permitiram verificar que o sistema implementado na FPGA funciona como na simulação, sendo que com os VDMAs só foram analisados os *links* individualmente.

Este subsistema é o último apresentado e no próximo capítulo vai ser demonstrada como todos os subsistemas apresentado integram uns com os outros e como interagem com o subsistema que envia para o SDI.

Capítulo 8

Sistema completo

Neste capítulo é apresentada a integração do sistema desenvolvido com os módulos que enviam para os *links* SDI e os restantes que são necessários para implementar o sistema na FPGA utilizada. É também abordado o subsistema secundário *Configuration* que como já foi mencionado, é responsável por distribuir as configurações pelos restantes módulos.

8.1 Subsistema *Configuration*

Este subsistema utiliza o protocolo *AXI4-lite m1_axicrl* para fazer uma interface entre *hardware* e *software* programável ao registo, o que permite configurar 4 registos de 32 bit que depois são distribuídos pelos restantes módulos (figura 8.1). Quando são distribuídos pelos restantes módulos, apenas são utilizados os 16 bit menos significativos dos registos com a resolução horizontal, com a vertical e com o *frame rate*. Para a saída *int_fra* apenas é utilizado o bit menos significativo do registo que contem a informação para entregar a esta saída. As saídas são atualizadas quando existe um *reset* do sistema.

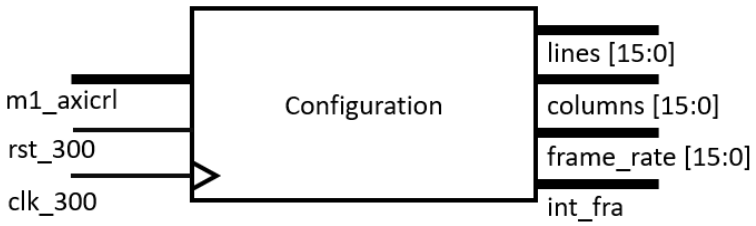


Figura 8.1: Diagrama de blocos do subsistema *Configuration*.

8.2 Integração do sistema

Após serem apresentados todos os subsistemas desenvolvidos ao longo do projeto em capítulos anteriores, nesta secção é apresentada a integração desses sistemas com os restantes módulo necessários. Os subsistemas podem ser agrupados e vistos como um único bloco que representa um sistema independente na figura 8.2 pelo blobo *UHD Test Pattern Generator*. Este apresenta duas interfaces *AXI4-lite*, uma para configurar o IP da Xilinx *Vídeo Test Pattern Generator* (interface *m0_axicrl*) e outra para configurar o subsistema *Config* (interface *m1_axicrl*). Apresenta também uma entrada *clk_300* que recebe o *clock* com frequência de 300 MHz e uma *rst_300* que recebe o *reset* síncrono com este *clock* a distribuir pelos diversos subsistemas. Relativamente às saídas, este bloco apresenta 4 interfaces *AXI4-stream* para enviar para o bloco *SDI TX* a informação dividida por 4 *links*.

O bloco *Micro Blaze* representa o *soft processor* da FPGA [43], que atribui as configurações aos registos de configuração por *AXI4-lite* para o bloco *UHD Test Pattern Generator* e para o *SDI TX*. Este também é responsável pela configuração do *Programable clock* por I2C. Este *clock* é programado para 148.5 MHz, pois é a frequência de *clock* que é necessária no bloco *SDI TX*. Este consegue gerar divisões e multiplicações desta frequência automaticamente, dependendo do tipo de protocolo configurado (SDI 3G, SDI 6G, SDI 12G ...). As configurações necessárias para este bloco chegam pela interface *m2_axisclr*. Por último o bloco *Default Clock* é o gerador de *clock* que está na placa e distribui o *clock* com frequência de 300 MHz para o bloco *UHD Test Pattern Generator* e para o *SDI TX*. Este último recebe dois *clocks* com frequências diferentes, em que um serve para receber os sinais das *AXI4-streams* e o outro para enviar os sinais pelas interfaces *SDI*.

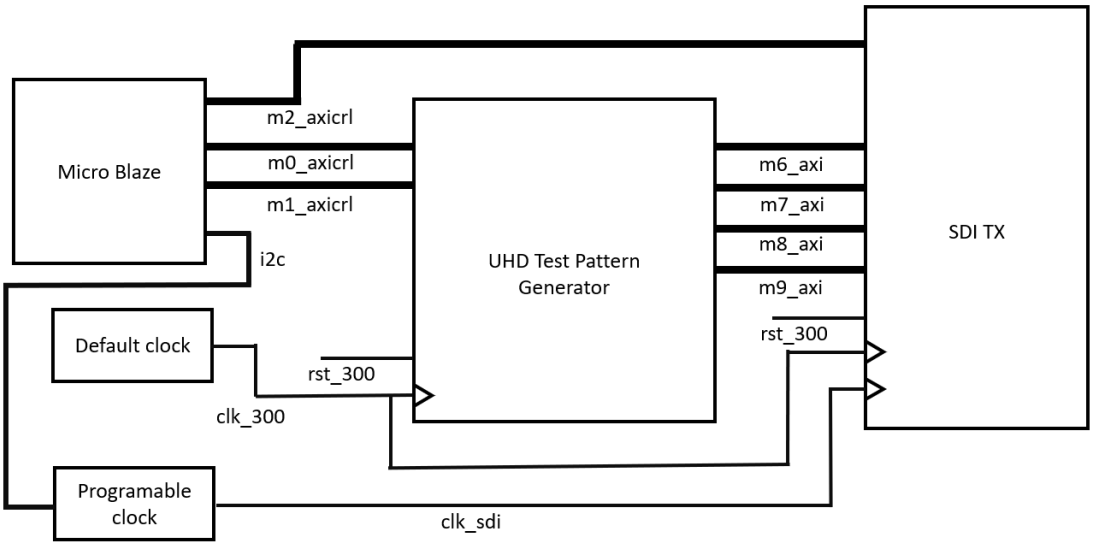


Figura 8.2: Integração do *UHD Test Pattern Generator* com os restantes blocos do sistema.

8.3 Módulos responsáveis pelo bloco SDI Tx

O bloco SDI TX é composto por dois módulos IP da Xilinx. Um é o *SMPTE UHD-SDI RX SUBSYSTEM* que é responsável por receber os 4 links de *AXI4-stream* vindos do *UHD Test Pattern Generator* e transforma-los nos *data streams* mencionados na secção 2.7 [34] (para uma interface *quad link* existem 32 *data streams*, 8 para cada *link*, mas saem deste módulo multiplexadas numa única *stream* por link). O outro módulo é o UHD-SDI GT [35] sendo responsável pelo *transceiver* físico mencionado na secção 3.7 do capítulo 3, que passa os dados das *data streams* multiplexadas, de paralelo para série (par diferencial).

O problema que impede o funcionamento deste conjunto de módulos representado pelo bloco *SDI TX* da figura 8.2, e que impede o teste em condições reais está no módulo *SMPTE UHD-SDI RX SUBSYSTEM*. Este não se consegue configurar corretamente, o que impossibilita a existência de sinal nas saídas físicas de SDI, o que resulta na impossibilidade de observar o sinal dos *links* SDI, por exemplo em ecrãs. .

8.4 Teste

O funcionamento incorreto para altas resoluções dos módulo responsáveis pelo bloco SDI TX, impossibilita o teste em condições reais. No entanto, foram tomadas duas alternativas para testar o sistema implementado neste projeto, novamente com os VDMAs, mas agora com algumas diferenças.

Os IP VDMAs apresentam a limitação de apenas permitirem guardar 32 *frames*, mas a primeira alternativa consistiu em alterar a aplicação de *software* para colocar os VDMAs a guardar *frames* de forma cíclica. Isto não permite guardar um maior número de *frames*, mas permite estar sempre a sobrepor até existir um *breakpoint* e não parar após as 32 primeiras. Nesse momento os VDMAs vão conter 32 *frames* de um certo instante da *stream*.

A segunda alternativa, que é melhor que a primeira, consistiu na utilização de um IP desenvolvido noutro projeto na empresa que emula o comportamento do VDMA, mas permite guardar mais do que 32 *frames*. Este IP permite uma análise dum intervalo de tempo de vídeo mais alargado.

Com esta ultima metodologia de teste foi possível verificar se o retângulo preto está síncrono com o vídeo e *timecode* num determinado instante tempo da *stream*.

8.5 Resultados

Na figura 8.3 é apresentado uma *frame* com o padrão *"Color Bars"*, resultante da segunda alternativa de teste apresentada em cima. Esta é a *frame* número 11, do segundo 19, numa configuração 1080p 60 *fps*, ou seja, é a *frame* número 1151 (60 x 19+11). Como se verifica na figura, e se afirma no capítulo 5, uma *frame* de um segundo ímpar apresenta o retângulo preto ativo.

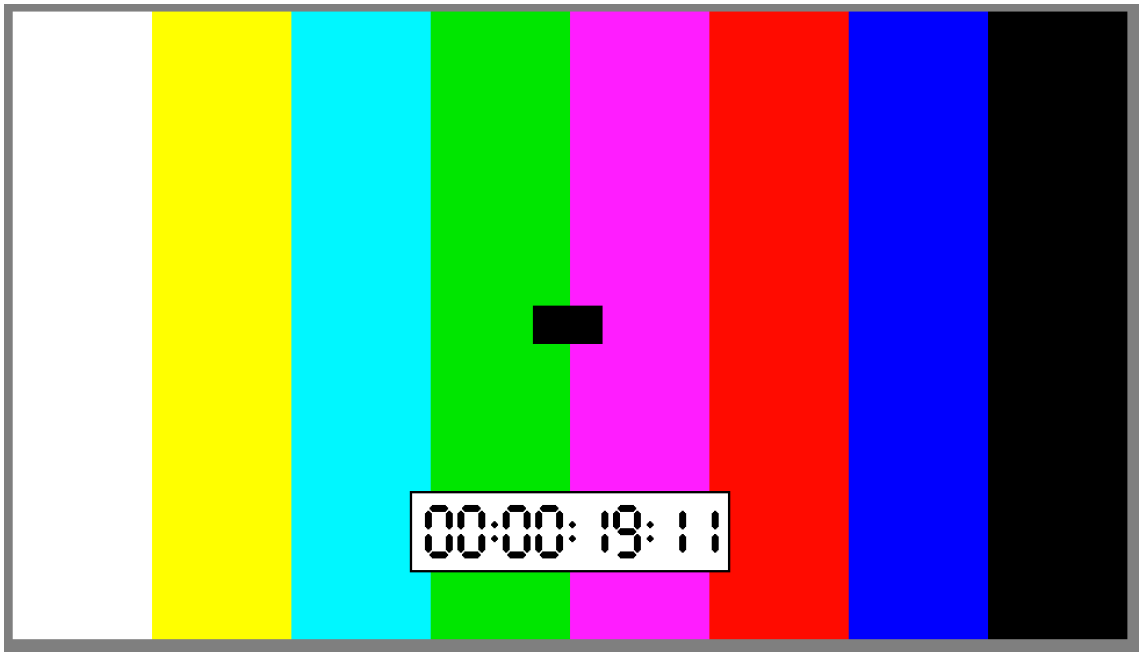


Figura 8.3: *Frame* 1080p com o padrão *"Color Bars"* retirada do VDMA do *link* 1

Se o sistema for configurado para trabalhar com a resolução 1080p, são guardados nos 4 VDMAs *frames* com a resolução 960 x 540. Cada *frame* neste formato ocupa cerca de 1.3 MB (fórmula 8.1), e como este IP desenvolvido noutro projeto permite guardar 16 GB em memória é possível guardar cerca de 12000 *frames* (fórmula 8.2), se apenas se incluir um VDMA em um dos *links* do *design* e nos outros não se guardar a informação.

$$Tamanho\,da\,frame\,(MB) = resolução\,horizontal \times resolução\,vertical \times \frac{20}{8 \times 1000} \tag{8.1}$$

$$\text{Número}\,de\,frames\,(MB) = \frac{Tamanho\,da\,frame\,(MB)}{16\,(GB)} \tag{8.2}$$

8.6 Recursos da FPGA utilizados

Após implementação do sistema completo em FPGA, os recursos utilizados relativamente ao bloco *UHD Test Pattern Generator* da figura 8.2 são os apresentados na tabela 8.1.

Tabela 8.1: Recursos utilizados para o bloco *UHD Test Pattern Generator*.

Módulo	**LUTS**	**FF**	**BRAMs**	**DSP**
Test Pattern Generator	7631	7320	12	82
YUV Filter	0	0	0	0
Black Box	374	56	0	0
White Box	485	31	0	0
Numbers	253	8164	0	0
16-bit Divider	82	116	0	0
AXI Data FIFO	95	69	36.5	0
Organizer	851	176	0	0
Distributor	1	0	0	0
Config	56	170	0	0
Total	**10359**	**16657**	**158**	**82**
Total da FPGA	**1182240**	**2364480**	**2160**	**6840**
Percentagem de utilização (%)	**0.9**	**0.7**	**7.3**	**1.2**

Pela análise da tabela, é possível verificar que os módulos que utilizam mais LUTS e FF são o IP da Xilinx *Test Pattern Generator* e o *Numbers*. Este último apresenta muitos FF, pois têm de registar o valor enumeras constantes com muita lógica associada para guardar as posições dos segmentos dos números. Módulos como o *Black Box*, *White Box*, o e *Numbers* apresentam menos FF, pois utilizam menos lógica associada às constantes para guardarem as posições dos objetos. O *Orgranizer* apresenta um número considerável de LUTS, mas poucos FF, pois este regista informação com pouca lógica associada. Os módulos *YUV Filter* e o *Distributor* não registam informação, e este último só realiza uma operação lógica, o que faz que praticamente não utilizem recursos. Por último, o IP *AXI Data FIFO* utiliza algumas BRAMs para armazenar od dados das FIFOs.

Como se pode verificar na tabela acima o projeto necessita de poucos recursos da FPGA, e as percentagens de utilização da mesma são bastante baixas, pois a *Virtex UltraScale+ XCVU9P-L2FSGD2104E* é uma FPGA *high range* [47].

Existem produtos de vídeo no mercado para a resolução 8K que utilizam a FPGA *Kintex UltraScale+ XCKU040*. Esta FPGA apresenta 242400 LUTs, 484800 FF, 600 BRAMs e 1920 DSP o que resulta em percentagens de utilização de 4.3%, 3.4%, 26.3% e 4.3% respetivamente, para o conjunto de módulos do sistema *UHD Test Pattern Generator*. O que permitia, do ponto de vista de utilização de recursos, que este projeto também poderia ser implementado neste tipo de FPGA.

Este capítulo finaliza a descrição de todos os subsistemas integrados com os restantes módulos necessários para implementar o projeto na FPGA. Apresenta também o teste alternativo ao teste em condições reais e a utilização de recursos da FPGA utilizada.

Para finalizar este relatório relativo ao projeto desenvolvido neste projeto, no próximo capítulo vão ser abordadas as conclusões e trabalho futuro.

Capítulo 9

Conclusão

Neste último capítulo da dissertação são abordadas as conclusões relativas à realização deste projeto. Primeiro é realizado um sumário do projeto incluindo conclusões relativas ao cumprimento de requisitos e objetivos. Por último é apresentada uma secção que aborda o trabalho futuro, ou seja, o que poderia tornar este projeto mais completo e em que aspetos este projeto contribui ou pode vir a contribuir para outros projetos em desenvolvimento na MOG.

9.1 Sumário

Em suma, o sistema desenvolvido neste projeto foi concebido com o objetivo de gerar padrões de vídeo de 8K 60 *fps* para a interface SDI 12G. No entanto, este também pode ser utilizado para testar interfaces SDI 3G e SDI HD (1.5 G) quando configurado para funcionar com as resoluções 4K e 1080p, respetivamente.

Os requisitos do projeto foram todos cumpridos, de tal forma que o sistema pode ser utilizado com diversas resoluções e *frame rates* que podem ser configurados por uma aplicação de *software*. Para além disso, o sistema está desenhado de forma a facilitar a adição de novas resoluções e *frame rates*, pois na indústria do setor surgem frequentemente novos formatos de vídeo.

Quanto aos objetivos, estes foram todos cumpridos com exceção do teste em condições reais de utilização devido a um problema na configuração dos módulos da Xilinx responsáveis por converter a informação que chega por *AXI4-stream* para informação que possa ser enviada pela norma SDI. No entanto, foram utilizadas metodologias alternativas de teste que conseguem comprovar o funcionamento dos diferentes subsistemas desenvolvidos ao longo do projeto.

Este projeto contribui também de uma maneira positiva para outros projetos em desenvolvimento na empresa neste momento. Os diferentes subsistemas implementados são independentes uns dos outros e podem ser utilizados em separado, incluindo os seus IPs noutros projetos. É o caso do subsistema responsável por implementar o método 2SI que apresenta grande utilidade para outros projetos da empresa, pois implementa um método de *multilink*. Este IP pode ser utilizado, por exemplo num projeto que não seja um gerador de padrões de vídeo, mas que necessite de dividir uma *stream* em quatro. Um módulo que implemente *multilink* é importante, porque esta

é a única forma de transmitir 8K 60 *fps* em SDI no momento, e existem projetos na empresa que necessitam de utilizar esta técnica.

O *testbench* implementado para este projeto também pode ser usado para testar outros projetos de vídeo que utilizem *AXI4-stream*, que estão em desenvolvimento na MOG.

9.2 Trabalho futuro

Relativamente ao trabalho futuro, o teste em condições reais de utilização é algo que ficou por realizar, e quando o problema na configuração do módulo *SMPTE UHD-SDI RX SUBSYSTEM* estiver resolvido, o sistema está pronto para ser submetido a este teste.

Um dos aspetos que pode ser ainda abordado para um teste mais completo do sistema é a realização de testes *cross check*. Este teste pode consistir num *testbench* que realize a comparação das componentes dos pixeis resultantes de uma *frame* à saída do sistema implementado com os valores que são esperados que eles tomem. Isto permitia verificar se todos os bit de uma determinada *frame* tomam os valores esperados.

Na continuação deste projeto existe também o interesse de adicionar um subsistema que implemente um gerador de áudio a trabalhar em simultâneo com o gerador de padrões de vídeo. A ideia passa por permitir testar o áudio e o vídeo e a sua simultaneidade, através do subsistema *Black Box*.

Este projeto foi desenvolvido de forma a poder ser alterado para suportar *frame rates* e resoluções maiores que possam vir a surgir no futuro. No caso do suporte para trabalhar com novas normas SDI como é o caso do SDI 24G, este pode ser atingido através de alterações nos tamanhos dos barramentos de dados *AXI-stream* conciliadas com o natural desenvolvimento do IP *Vídeo Test Pattern Generator* por parte da Xilinx para resoluções cada vez maiores e um aumento do número de amostragens por ciclo de relógio.

Anexo A

Caraterísticas da diferentes normas SDI

Na figura apresentada na próxima página são apresentadas algumas características das diferentes normas SDI.

Standard:	270Mb/s	1.5Gb/s	3Gb/s	6Gb/s	12Gb/s	24Gb/s
	ST 259M	ST 292-1	ST 424	Proposed ST 2081-1	Proposed ST 2082-1	Proposed ST 2083-1
Coding:	Scrambled NRZI	Scrambled NRZI	Scrambled NRZI	Scrambled NRZI	Scrambled NRZI	Scrambled NRZI
Amplitude:	800mV ±10%	800mV ±10%	800mV ±10%	800mV ±10%	800mV ±10%	800mV ±10%
DC Offset:	0.0V ±0.5V	0.0V ±0.5V	0.0V ±0.5V	0.0V ±0.5V	0.0V ±0.5V	0.0V ±0.5V
Rise-/fall time:	400ps ...1.5ns	≤ 270ps	≤ 135ps	≤ 80ps	≤ 45ps	≤ 28ps
Δ Rise-/fall time:	≤ 500ps	≤ 100ps	≤ 50ps	≤ 35ps	≤ 18ps	≤ 8ps
Over- / under-shoot:	10% of the amplitude	10% of the amplitude	10% of the amplitude	10% of the amplitude	10% of the amplitude	10% of the amplitude
Timing Jitter:	<0.2UI up to 10Hz	<1UI up to 10Hz	< 2 UI up to 10Hz	<2 UI up to 10Hz	< 2 UI up to 10Hz	< 2 UI up to 10Hz
Alignment Jitter: Lower edge kHz> 1/10 clock rate	<0.2UI 1KHz to 27Mhz	<0.2UI 100KHz to 150Mhz	< 0.3 UI 100KHz to 300Mhz	< 0.3 UI 100KHz to 600Mhz	< 0.3 UI 100KHz to 1200Mhz	< 0.3 UI 100KHz to 2400MHz
Return Loss	<15 dB - 5 MHz to 270MHz	<15 dB - 5 MHz to 1.5GHz	<15 dB - 5 MHz to 1.5GHz; <10 dB -1.5Ghz to 3GHz	<15 dB - 5 MHz to 1.5GHz; <10 dB -1.5Ghz to 4.5GHz; <8 dB – 4.5GHz to 6GHz	<15 dB - 5 MHz to 1.5GHz; <10 dB -1.5Ghz to 4.5GHz; <8 dB – 4.5GHz to 6GHz <6 dB – 6GHz to 12GHz	<15 dB - 5 MHz to 1.5GHz; <10 dB -1.5Ghz to 4.5GHz; <8 dB – 4.5GHz to 6GHz <6 dB – 6GHz to 12GHz <4 dB – 12GHz to 24GHz
75 Ω Coaxial Cable length (Point-to-point)	400m+	300m	200m	100m+	60m+	≤40m

Anexo B

Equações de conversão RGB para YCbCr

Equações gerais para a conversão de RGB para YCbCr [23]:

Equação 1: Y = (0.257 x R) + (0.504 x G) + (0.098 x B) + 16
Equação 2: Cb = - (0.148 x R) - (0.291 x G) + (0.439 x B) + 128
Equação 3: Cr = (0.439 x R) - (0.368 x G) - (0.071 x B) + 128

Equações gerais para a conversão de YCrCb para RGB:

Equação 5: R = 1.164 x (Y - 16) + 1.596 x (Cr - 128)
Equação 5: G = 1.164 x (Y - 16) - 0.813 x (Cr - 128) - 0.391 x (Cb - 128)
Equação 6: B = 1.164 x (Y - 16) + 2.018 x (Cb - 128)

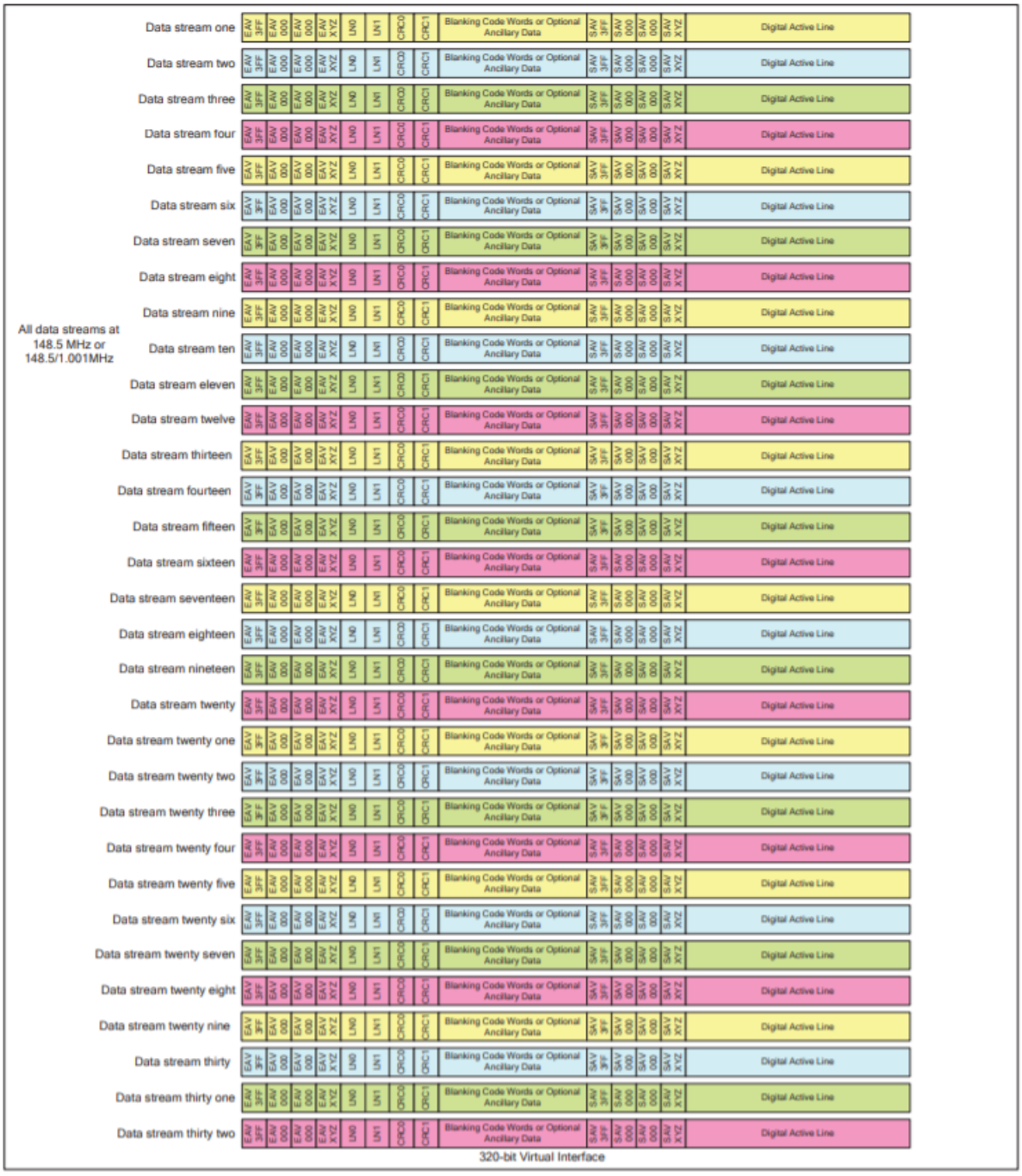
Na forma de matriz:

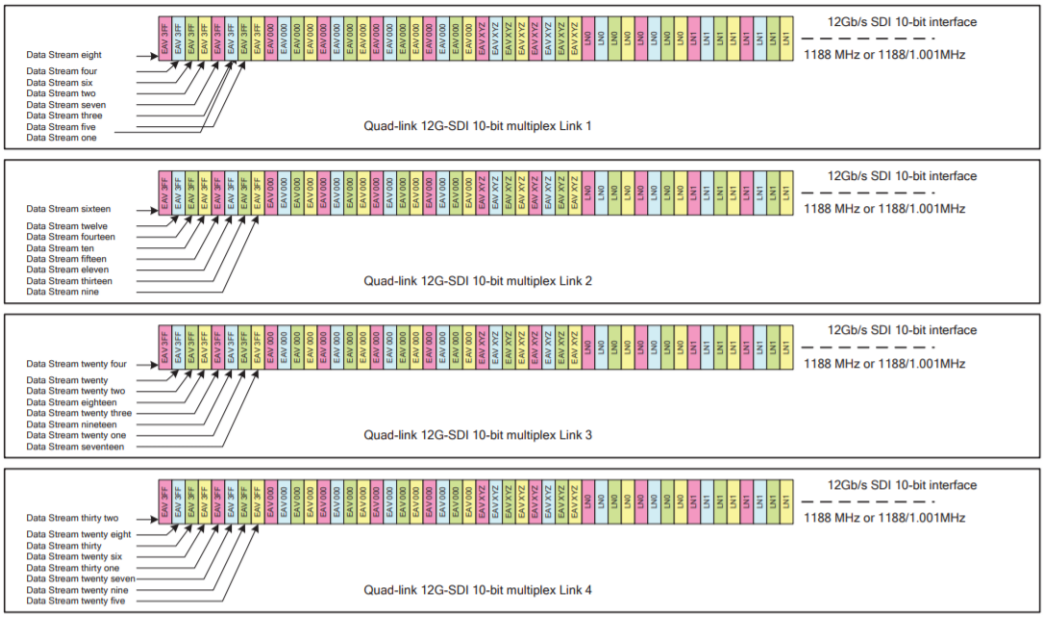
$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Anexo C

Mapeamento *quad-link* 12G-SDI 10-*bit*

Nas figuras das próximas páginas são apresentadas as 32 *data streams* existem para uma interface *quad-link* 12G-SDI e a sua multiplicação em 4 *streams* para serem enviadas pelos 4 *links*.


All data streams at 148.5 MHz or 148.5/1.001MHz
Data stream one
Data stream two
Data stream three
Data stream four
Data stream five
Data stream six
Data stream seven
Data stream eight
Data stream nine
Data stream ten
Data stream eleven
Data stream twelve
Data stream thirteen
Data stream fourteen
Data stream fifteen
Data stream sixteen
Data stream seventeen
Data stream eighteen
Data stream nineteen
Data stream twenty
Data stream twenty one
Data stream twenty two
Data stream twenty three
Data stream twenty four
Data stream twenty five
Data stream twenty six
Data stream twenty seven
Data stream twenty eight
Data stream twenty nine
Data stream thirty
Data stream thirty one
Data stream thirty two
EAV 3FF EAV 000 EAV 000 EAV XYZ LN0 LN1 CRC0 CRC1
Blanking Code Words or Optional Ancillary Data
SAV 3FF SAV 000 SAV 000 SAV XYZ
Digital Active Line
320-bit Virtual Interface


Data Stream eight
Data Stream four
Data Stream six
Data Stream two
Data Stream seven
Data Stream three
Data Stream five
Data Stream one
12Gb/s SDI 10-bit interface
1188 MHz or 1188/1.001MHz
Quad-link 12G-SDI 10-bit multiplex Link 1
Data Stream sixteen
Data Stream twelve
Data Stream fourteen
Data Stream ten
Data Stream fifteen
Data Stream eleven
Data Stream thirteen
Data Stream nine
12Gb/s SDI 10-bit interface
1188 MHz or 1188/1.001MHz
Quad-link 12G-SDI 10-bit multiplex Link 2
Data Stream twenty four
Data Stream twenty
Data Stream twenty two
Data Stream eighteen
Data Stream twenty three
Data Stream nineteen
Data Stream twenty one
Data Stream seventeen
12Gb/s SDI 10-bit interface
1188 MHz or 1188/1.001MHz
Quad-link 12G-SDI 10-bit multiplex Link 3
Data Stream thirty two
Data Stream twenty eight
Data Stream thirty
Data Stream twenty six
Data Stream thirty one
Data Stream twenty seven
Data Stream twenty nine
Data Stream twenty five
12Gb/s SDI 10-bit interface
1188 MHz or 1188/1.001MHz
Quad-link 12G-SDI 10-bit multiplex Link 4

Referências

[1] SAMSUNG. Tecnologia 4k, Março 2017. http://patrocinados.estadao.com.br/samsung/2017/03/31/porque-o-4k-e-a-tecnologia-que-veio-para-ficar/, Consultado a 10/12/2018.

[2] Nigel Seth-Smith John Hudson e Randy Conrod. UHD in a Hybrid SDI/IP World. *SMPTE Motion Imaging Journal (Volume: 125 , Issue: 2 , March 2016)*, October 2015. https://www.smpte.org/sites/default/files/users/user27446/Soiree%20CES%20-%204K%20-%20UHD_hybrid_SDI_IP_Imagine.pdf, Consultado a 10/12/2018.

[3] Roger Pink. 8K TVs: An Evolution of Resolution, Janeiro 2018. https://electronics360.globalspec.com/article/11037/8k-tvs-an-evolution-of-resolution, Consultado a 10/12/2018.

[4] Jungwook Wee Kyungwon Park Kiwon Kwon Seulki Song, Youngsu Ryu. Design and Implementation of 8K UHD Encapsulation Method for Efficient Transmission and Reception based on MMT. *KSII Transactions on Internet & Information Systems, Vol. 12 Issue 2, p860-872. 13p.*, Fevereiro 2018. http://itiis.org/digital-library/manuscript/1934.

[5] José Sá. Study on the FPGA implementation of the conversion of uncompressed High-Definition video signals for distribution over IP networks. *U. PORTO - Repositório Aberto*, Julho 2016. https://repositorio-aberto.up.pt/bitstream/10216/89271/2/170130.pdf.

[6] Takayuki Yamashita Michael Van Dorpe Masayuki Miyazaki, Tsuyoshi Sakiyama. 8K-TICO Codec for Miniaturized and Simplified UHDTV Production Systems. *SMPTE 2017 Annual Technical Conference and Exhibition*. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8281417.

[7] VIdeo Products Inc. Uses and Benefits of Video Test Pattern Generators, Dezembro 2019. https://www.montest.com/technical-articles/uses-benefits-of-video-test-pattern-generator-45.html, Consultado a 01/02/2019.

[8] Wikipédia. Resolução de imagem, Maio 2019. https://en.wikipedia.org/wiki/YCbCr, Consultado a 03/02/2019.

[9] idearocket. Understanding Aspect Ratio: A Quick Guide to Online Video Formats, Junho 2017. https://idearocketanimation.com/16662-video-aspect-ratio-guide/?utm_referrer=https%3A%2F%2Fwww.google.com%2F, Consultado a 03/02/2019.

[10] frame.io. Timecode. https://workflow.frame.io/guide/timecode, Consultado a 31/05/2019.

[11] Marcelo Ruiz. Entrelaçado ou progressivo? Eis a questão! https://olharmultimidiatec.blogspot.com/2012/04/entrelacado-ou-progressivo-eis-questao.html, Consultado a 31/05/2019.

[12] Margaret Rouse. Serial Digital Interface (SDI). https://searchnetworking.techtarget.com/definition/Serial-Digital-Interface, Consultado a 27/01/2019.

[13] John Hudson. Uhd-sdi standards overview – towards a hierarchy of sdi data rates. Relatório técnico, Semtech Corporation – Gennum Products Group. https://www.smpte.org/sites/default/files/u388/Semtech%20PDF%20Presentation.pdf, Consultado a 27/01/2019.

[14] OPTCORE. Introduction to SDI, HD-SDI, 3G-SDI, 6G-SDI, 12G-SDI and 24G-SDI, Novembro 2017. https://www.optcore.net/introduction-to-sdi/, Consultado a 27/01/2019.

[15] THE SOCIETY OF MOTION PICTURE & TELEVISION ENGINEER. ST 297-2:2017 - SMPTE Standard - Multi-Link and Multi-Channel 1.5G, 3G, 6G and 12G-SDI Using CWDM. Relatório técnico, Janeiro 2017.

[16] Scott Barella. The transition from uncompressed SDI to IP video. Relatório técnico, Utha Scientific. http://www.ste-ca.org/images/The_Transition_from_SDI_to_IP_VIDEO_Utah_6C.pdf, Consultado a 27/01/2019.

[17] Pitivi Video Editor. Understanding codecs and containers. http://www.pitivi.org/manual/codecscontainers.html, Consultado a 01/02/2019.

[18] Tecktronix. UHD-4K SQD vs 2SI Square Division versus 2 Sample Interleave. Relatório técnico, Abril 2018. https://www.abacanto.net/wp-content/uploads/2018/04/TEKTRONIX-TIPSTRICKS-148-UHD-4K-SQD-vs-2SI-ES.pdf, Consultado a 01/02/2019.

[19] Delcast. 4K Video - The road towards 4K, 2019. https://www.deltacast.tv/technologies/4k-video, Consultado a 01/02/2019.

[20] Ross. UHDTV-1 (4K) Production. Relatório técnico. https://www.live-production.tv/sites/default/files/acuity_4K_production.pdf, Consultado a 03/02/2019.

[21] Makarand Tapaswi. Why the RGB to YCbCr, Junho 2009. https://makarandtapaswi.wordpress.com/2009/07/20/why-the-rgb-to-ycbcr/, Consultado a 01/02/2019.

[22] SMPTE. *UHDTV Ecosystem Study Group Report*. IEEE Xplore, 2014. https://www.smpte.org/sites/default/files/Study%20Group%20On%20High-Dynamic-Range-HDR-Ecosystem.pdf, Consultado a 03/02/2019.

[23] Md. Golam Rabbani Mirza Rehenuma Tabassum Alim Ul Gias Md. Mostafa Kamal Hossain Muhammad Muctadir Asif Khan Shakir Asif Imran Shah Mostafa Khaled, Md. Saiful Islam

e Saiful Islam. Combinatorial Color Space Models for Skin Detection in Sub-continental Human Images. Relatório técnico, Combinatorial Color Space Models for Skin Detection in Sub-continental Human Images, 2009. https://www.researchgate.net/publication/221365117_Combinatorial_Color_Space_Models_for_Skin_Detection_in_Sub-continental_Human_Images.

[24] Wikipédia. YCbCr, Janeiro 2019. https://pt.wikipedia.org/wiki/Resolu%C3%A7%C3%A3o_de_imagem, Consultado a 30/05/2019.

[25] Wikipédia. Chroma subsampling, Janeiro 2019. https://en.wikipedia.org/wiki/Chroma_subsampling, Consultado a 01/02/2019.

[26] Louis Frenzel. *Payload Identification Codes for Serial Digital Interfaces (SMPTE ST 352-2010)*. 2011.

[27] SMPTE. *12G-SDI Bit-Serial Interfaces — Overview for the SMPTE ST 2082 Document Suite (SMPTE OV 2082-0)*. IEEE Xplore, 2016.

[28] SMPTE. *2160-line Source Image and Ancillary Data Mapping for 12G-SDI (SMPTE ST 2082-10)*. IEEE Xplore, 2015.

[29] SMPTE. *12 Gb/s Signal/Data Serial Interface — Electrical (SMPTE ST 2082-1:)*. IEEE Xplore, 2015.

[30] SMPTE. *4320-line and 2160-line Source Image and Ancillary Data Mapping for Quad-link 12G-SDI (SMPTE ST 2082-12)*. IEEE Xplore, 2016.

[31] ARM. AMBA AXI4-Stream Protocol. Relatório técnico, 2012. https://static.docs.arm.com/ihi0051/a/IHI0051A_amba4_axi4_stream_v1_0_protocol_spec.pdf Consultado a 18/06/2019.

[32] Lauri's blog. Arbitrary data streams. https://lauri.xn--vsandi-pxa.com/hdl/zynq/axi-stream.html, Consultado a 31/05/2019.

[33] Xilinx. *Video Test Pattern Generator v8.0*. Abril 2018. https://www.xilinx.com/support/documentation/ip_documentation/v_tpg/v8_0/pg103-v-tpg.pdf. *Datasheet*.

[34] Xilinx. *SMPTE UHD-SDI Transmitter Subsystem v2.0*. Maio 2017. https://www.xilinx.com/support/documentation/ip_documentation/v_smpte_uhdsdi_tx_ss/v2_0/pg289-v-smpte-uhdsdi-tx-ss.pdf. *Datasheet*.

[35] Xilinx. *SMPTE UHD-SDI Receiver Subsystem v2.0*. Dezembro 2018. https://www.xilinx.com/support/documentation/ip_documentation/v_smpte_uhdsdi_rx_ss/v2_0/pg290-v-smpte-uhdsdi-rx-ss.pdf. *Datasheet*.

[36] Xilinx. *Video Broadcaster v1.0*. Outubro 2012. https://www.xilinx.com/support/documentation/ip_documentation/axis_vbroadcaster/v1_00_a/ds880-axis-vbroadcaster.pdf. *Datasheet*.

[37] Xilinx. *Integrated Logic Analyzer v6.2*. Abril 2016. https://www.xilinx.com/support/documentation/ip_documentation/ila/v6_1/pg172-ila.pdf. *Datasheet*.

[38] Xilinx. *AXI Video Direct Memory Access v6.2*. Novembro 2016. https://www.xilinx.com/support/documentation/ip_documentation/axi_vdma/v6_2/pg020_axi_vdma.pdf. *Datasheet*.

[39] Xilinx. *AXI4-Stream FIFO v4.1*. Abril 2016. https://www.xilinx.com/support/documentation/ip_documentation/axi_fifo_mm_s/v4_1/pg080-axi-fifo-mm-s.pdf, *Datasheet*.

[40] Omnitec. https://www.omnitek.tv/, Consultado a 12/02/2019.

[41] SMPTE. *TICO Lightwhigth Codec Used in IP Networked or in SDI Infrastructures*. IEEE Xplore, 2016.

[42] elecard. Yuv viwer. https://www.elecard.com/products/video-analysis/yuv-viewer, Consultado a 11/03/2019.

[43] Xilinx. Microblaze processor quick start guide. https://www.xilinx.com/support/documentation/quick_start/microblaze-quick-start-guide.pdf. *Datasheet*.

[44] Texas Instruments. *LMH1297 12G UHD-SDI Bidirectional I/O With Integrated Reclocker*. 2017 Outubro. http://www.ti.com/lit/ds/symlink/lmh1297.pdf. *Datasheet*.

[45] Xilinx. Xilinx virtex ultrascale+ fpga vcu1525 acceleration development kit. https://www.xilinx.com/products/boards-and-kits/vcu1525-a.htm, Consultado a 29/05/2019.

[46] Xilinx. Vcu1525 reconfigurable acceleration platform, Março 2019. https://www.xilinx.com/support/documentation/boards_and_kits/vcu1525/ug1268-vcu1525-reconfig-accel-platform.pdf. *Datasheet*.

[47] Xilinx. *UltraScale Architecture and Product Data Sheet: Overview v3.8*. Maio 2019. https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf. *Datasheet*.